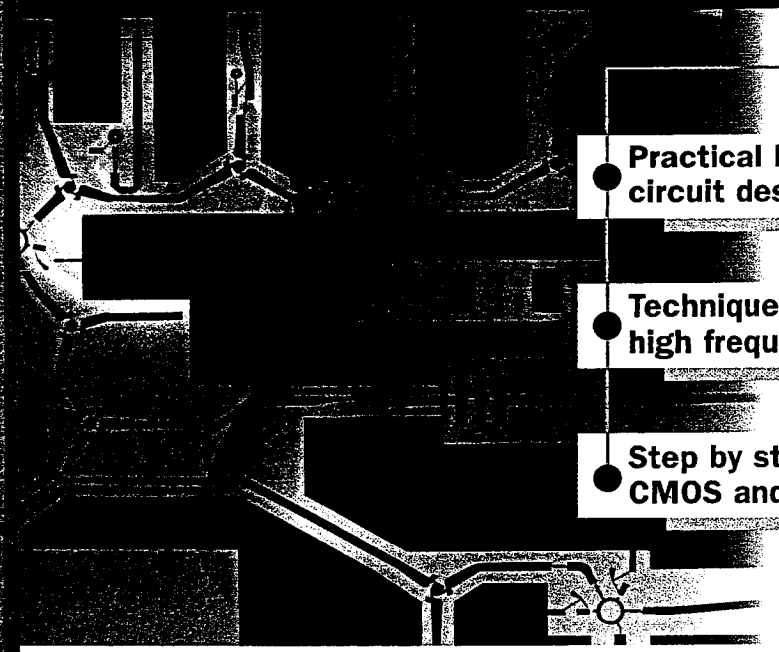
McGraw-Hill
PROFESSIONAL
ENGINEERING



- Practical layout reference for circuit designers and mask designers
- Techniques for matching, noise, high frequency issues and more
- Step by step case studies detailing CMOS and bipolar RFIC

IC Mask Design

ESSENTIAL LAYOUT TECHNIQUES

Christopher Saint / Judy Saint

Cataloging-in-Publication Data is on file with the Library of Congress

McGraw-Hill

A Division of The McGraw-Hill Companies

1 2 3 4 5 6 7 8 9 0 DOC/DOC 0 8 7 6 5 4 3 2

ISBN 0-07-138996-2

The sponsoring editor for this book was Stephen S. Chapman, the editing supervisor was Stephen M. Smith, and the production supervisor was Sherri Souffrance. It was set in Times Roman by TopDesk Publishers' Group.

Printed and bound by R. R. Donnelley & Sons Company.

McGraw-Hill books are available at special quantity discounts to use as premiums and sales promotions, or for use in corporate training programs. For more information, please write to the Director of Special Sales, McGraw-Hill Professional, Two Penn Plaza, New York, NY 10121-2298. Or contact your local bookstore.

This book is printed on recycled, acid-free paper containing a minimum of 50% recycled, de-inked fiber.

IC Mask Design

Essential Layout Techniques

Christopher Saint

Judy Saint

McGraw-Hill

New York | Chicago | San Francisco | Lisbon | London | Madrid

Mexico City | Milan | New Delhi | San Juan | Seoul

Singapore | Sydney | Toronto

CHAPTER 1

Digital Layout

Chapter Preview

Here's what you're going to see in this chapter:

- Close look at automated layout software
- Why automated layout only works with certain cells
- Knowing the circuit really does what it should
- How to know in advance if your floorplan choice is good
- Automated programs getting stuck
- Troubleshooting tips
- Which nets to wire first
- Which nets to wire by hand
- Techniques to guarantee rule-perfect layout
- Flowchart of digital layout procedures
- Lots of feedback loops
- How to keep the power moving through big cells
- Chicken or egg wiring and timing circle
- Did you really build what you designed?
- How to build quickie chips for testing

Opening Thoughts on Digital Layout

The majority of integrated circuits built today are large. I mean really huge CMOS digital chips. One chip might have literally millions of transistors in it. It's beyond any single mask designer's capabilities to lay out a chip like that by hand—in any reasonable time frame, at least. Consequently, the majority of large digital chips are laid out with the assistance of computer-aided tools.

Understanding how these automated digital layout tools operate allows you to develop skillful daily habits in your work—even in your analog work. If you understand how the software operates, you can lay out better circuits faster, compensate for software inadequacies, and steer clear of roadblocks before they happen.

Design Process

Let's build a digital chip. In this chapter, we will follow a design team as they progress from concept, through circuit testing, and finally to the actual gate placement and wiring of a digital chip, using a suite of software tools.

Let's start. It's the circuit designer's move first.

Verifying the Circuitry Logic

Circuit designers typically use languages called **VHDL** or **Verilog** to design their enormous digital circuits. VHDL stands for VHSIC (Very High Speed Integrated Circuits) Hardware Description Language, an IEEE standard since 1987. Verilog is another proprietary logic description language. We will use VHDL in our examples.

Circuit designers use the VHDL language to create a chip that exists first as only a database of numbers. The circuit designer's VHDL files are very C-like.[1] The files essentially say, for example, "I want a circuit function that adds two 16-bit numbers together." In this way, the VHDL files describe our microprocessor, our digital functions, or whatever functions we need.

These VHDL data files are then submitted to a computer simulator, which tests the chip circuitry while it is still in software form. The logic functions of the VHDL code run very quickly, much faster than a traditional transistor level SPICE simulation (but not as fast as the real silicon.)

The VHDL simulator needs to have process-specific software descriptions of each logic function it wants to use, such as rise time, fall time, gate propagation delays. This information, as well as other device parameters, is stored as a series of files that the VHDL simulator can access. Along with these electrical descriptions, there are also physical representations of each of the gates that the simulator and logic synthesizer can use. All of these files are collectively known as a **standard cell library** or **logic library.**

[1] The computer language, C.

VHDL Code Segment

```
architecture STRUCTURE of TEST is
 component and2x
 port(A,B,C,D: in std_ulogic := '1';
 Y: out std_ulogic);
 end component;
 constant VCC: std_ulogic := '1';
 signal T,Q: std_ulogic_vector(4 downto 0);
begin
T(0) <= VCC;
 A1: and2x port map(A=>Q(0), B=>Q(1),
Y=>T(2));
 A2: and2x port map(A=>Q(0), B=>Q(1),
C=>Q(2), D=>Q(3), Y=>T(4));
 Count <= Q;
```

The company that is supplying your silicon usually provides a standard cell library. Theoretically, you are given a library, which is perfect, and will stay perfect. However, updates to the library can occur quite frequently. Changes to the library can cause a once-perfect chip to stop functioning, especially if mistakes have been made in the updates. Updating a library mid-project is usually a bad move.

By looking at the results of these VHDL simulations, we can make adjustments to the circuitry before we commit the chip to actual silicon. This is a great saving.

Compiling a Netlist

Once the circuit designer has finished verifying his logic design, he will put his VHDL code through a **silicon compiler** or **logic synthesizer.** The compiler translates the high level C-like code into a file that contains all the required logic functions, as well as how they are to be connected to each other.

The file basically says, "In order to add two 16-bit numbers together, I need 25 gates and here's how they should be connected." In this way, all our logic functions are created and cross-referenced.

At this point, we know what gates we need, and we know how they must be eventually wired to each other. This file, called a **netlist,** will drive your automated layout tools.

Netlist Segment

```
module test ( in1, in2, out1);
 input in1,in2;
 output out1;
 wire \net1 , \net2 , \net3 ;
 AND2_2X U1 ( .Z(net1), .A(net2), .B(net3) );
 AND4_2X U2 ( .Z(net1), .A(net2), .B(net3),
.C(net2), .D(net1) );
endmodule
```

As the circuit designer begins to compile the VHDL code, he will control various switches. The switches control parameters such as area, power, and speed. Depending on chip requirements, the circuit designer might decide to compile the VHDL to prioritize only speed, only area, only power, or some specific combination of these interests. Results will vary depending on these priorities, so he inputs these choices to the compiler before it begins.

Drive Strength

The compiler can create nets that are extremely large. There may be hundreds of thousands of cells on one particular net, for instance. The more cells we have on a net, the more power we need to drive them. If we try to drive too many gates from a single source, we might overload our driving transistors. Our circuit will not work.

Therefore, before we can start layout, we need to modify the netlist to make sure that these large nets are adequately driven. To do this, we replace the cells that are driving the net with cells of identical logic function that have larger driving capability. Driving capability is referred to as the **drive strength,** or **fan out,** of the cell. The fan out number indicates how many devices a gate can drive. A driving gate can be any cell in a library.

For example, we might see that our cell library has 10 or 15 different sizes of inverters. These inverter selections might be referred to as *1x, 2x,* or *4x* inverters. These designations on the inverters refer to the drive strength of each inverter. Since a *1x* can typically drive two gates, a *2x* would drive four gates, and a *4x* would drive eight gates.

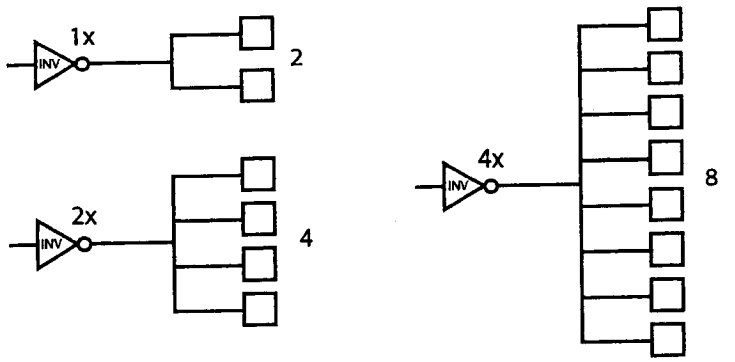


Figure 1–1. One inverter drives two loads, so a 2x drives 4 loads, and a 4x drives 8 loads.

You might wonder why we don't just build one huge gate to cover all circuit eventualities. We could do that. However, we would waste circuit area and burn more power than is necessary. The wisest technique is to be sure that you can drive what you need, and no more. So, during the compilation process, the compiler examines the number of gates on each net and adjusts the size of the gate driving each net accordingly. If the net is too large to be driven by our maximum drive strength device, the compiler will break the net into smaller sections that are easier to drive.

Buffer Cells

If the compiler breaks a large net into smaller, more easily driven sections, it will insert additional gates to drive each smaller newly created net. These extra gates were not part of the original logic. The circuit designer did not add them. You did not add them. The computer made the decision by itself.

These extra gates are called **buffer cells.** Buffer cells help drive gate and wiring capacitance. A buffer cell has no logic function associated with it. Whatever logic signal is fed into the buffer cell appears at its output.

In the next section, we will see an example of how the compiler uses these concepts to drive a large clock net.

Clock Tree Synthesis

Most digital circuits have a clock waveform that clicks away in the background. Every function is synchronized to that click. The wiring nets for this clock timing signal are called **clock nets.** A clock net is usually very large. Typically, the net connects to thousands of gates.

It is impossible to create a cell with enough drive strength to drive all the gates on a clock net, so we have to do some extra work to get the clock net to function. We split the clock net into smaller sections and add buffer cells, as mentioned previously. The net is split into a branching-out pattern, called a **clock tree.** Establishing the tree is called **clock tree synthesis.**

To illustrate how the tree concept works, let's look at a small example. Let's say a certain clock net has six gates on it, and the maximum drive strength our library offers is a fan out of only three. Therefore, we cannot expect one gate to drive the entire clock net. So, we break the net into two smaller sections, and drive each section separately.

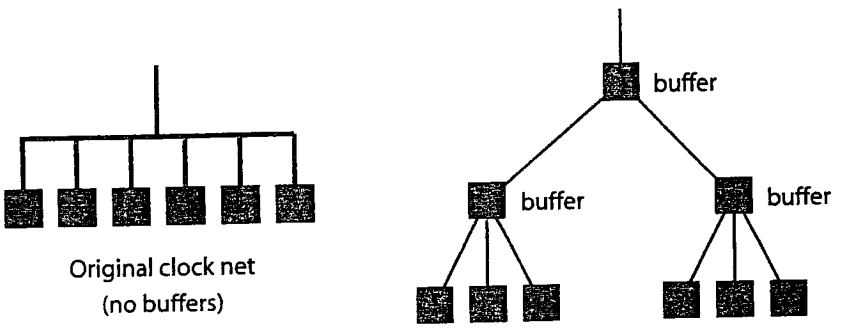


Figure 1–2. *Adding buffers to smaller sets of gates to help drive the signal.*

You can see in Figure 1–2, that the compiler added two lower level buffers to the circuit, one buffer to drive each set of three gates. The compiler also added another higher level buffer to drive the two lower level buffers. So, three extra cells have been added to our circuit.

If our clock net was even larger, the compiler would continue branching in this manner, splitting the net and adding additional buffer cells, each one driving no more than three others. You can see how this would form a very large tree with many levels.

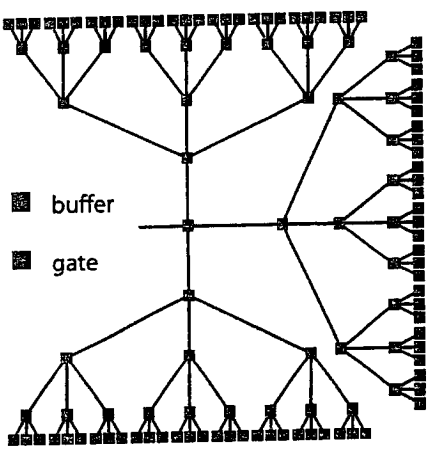


Figure 1–3. *Large nets are broken into many smaller sections that can more easily be driven.*

With a large number of added buffer cells, the extra cells will introduce extra delays that were not accounted for in the original simulations. Not only that, but other large nets may require this same sort of tree synthesis as well, adding even more buffer cells, also creating delays. Therefore, once the clock net is synthesized, and any other large fan out nets are buffered, we need to re-simulate our design using the compiled net list. Compiling creates a need to re-simulate. This sort of iteration is common in chip development. The good news is that it is not a never-ending story. At some point, you will have a finished netlist.

We are now ready to start the layout process. We begin with floorplanning.

Layout Process

We are now ready to use a suite, or package, of automated software tools called the **place and route tools.** Place and route tools cover the gamut of higher level and lower level software assistance leading to your final layout. As the name implies, these programs generally *place* the gates and *route* the wires, in addition to other helpful functions.

Floorplanning

The first piece of software we will use from the place and route tool suite is called the **floorplanning tool.** It will help you create areas of functionality on your chip, determine the connectivity between these areas, determine your I/O pad placements, and give you feedback on how easy your floorplan might be to wire. The floorplanning tool gets its connectivity and gate information based on the netlist file, created by the compiler software.

Let's follow the floorplanning tool in more detail, beginning with your initial decisions.

Block Placement

Typically, your chip will be divided into various functional areas. For example, if you are working on a large digital chip, there might be a microprocessor unit in your chip (MPU), perhaps a floating point unit (FPU), maybe a RAM block and a ROM block.

Where you locate each area of functionality is your decision, not the computer's. You might say, "Ok, all of the gates for the microprocessor I want in the bottom left hand corner. All the gates for the RAM I want in the top right corner." And so on. You will have a chance to change these decisions later, once you see how your decisions might affect your layout, particularly the wiring.

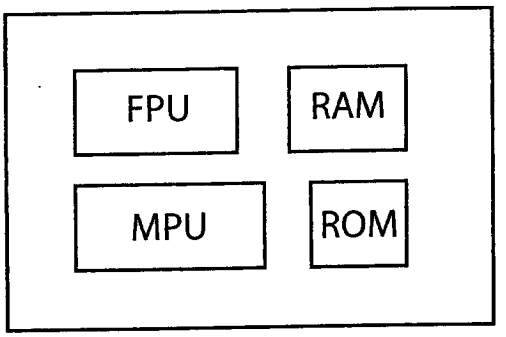


Figure 1–4. Chips have well-defined areas of functionality.

Gate Grouping

Once your areas of functionality are specified, the first task you would want to do is gather together, to some degree, the gates used in each block. You would not want FPU gates scattered throughout the ROM or RAM blocks, for example. Associated gates should all be located near each other.

The floorplanning tool begins by helping you gather your gates together. The exact placement of each gate is not determined at this point. We do not yet need this level of detail. Besides, we might be changing our block placement decisions at some later point. So general vicinity placement is good enough for now.

Block Level Connectivity

Next, your floorplanning tool will help you place the input and output (I/O) cells of your chip. For instance, you would want all the inputs that go to the FPU close to the FPU block in the corner. To help you with this, some tools will actually place the I/O cells in the appropriate areas automatically; other tools will provide graphic feedback for you based on your placement decisions.

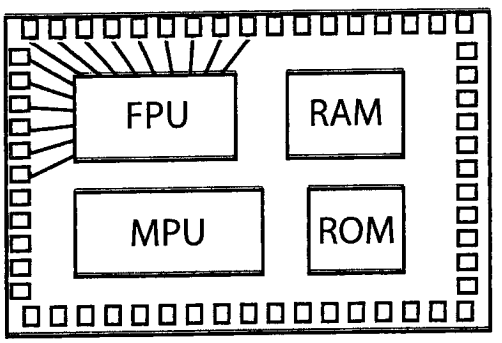


Figure 1–5. Inputs and outputs can be located near their appropriate cell block.

The floorplanning tool also shows basic wiring connections that must travel between blocks. It will show connections between the FPU and the RAM blocks, for example.

The automated software programs such as the simulator and place and route tools make choices a conscientious human with enough time would make, given the same information. (We hope.) However, you can see that constant human intervention and monitoring are essential.

The software never operates without human supervision (you). You have broadly defined where you want your high levels of functionality and your inputs and your outputs. You have predetermined some basic layout instructions for the software, depending on the chip specifications, the size of the final package in which they will be placed, the specific circuitry, and ultimately, on your understanding of how the software operates.

The tools never run completely by themselves. The human brain must oversee the working of the tools or the tools become useless.[2]

Using Flylines

Typically, the floorplanning tool will show you all the wiring lines coming from each block connecting to the I/O pads and to other blocks. All these myriad of wiring lines are what most tools call **rat's nests** or **flylines.**

As you click, drag, and resize blocks around your computer monitor, you will see all these wiring connections moving around in real time with your cursor.

As you drag your blocks around with your cursor, watch the lines. If the lines become badly crossed-over and generally messy, you know that it will be tough to wire the circuitry. If there are no cross-overs of the flylines, then it will be easy to wire.

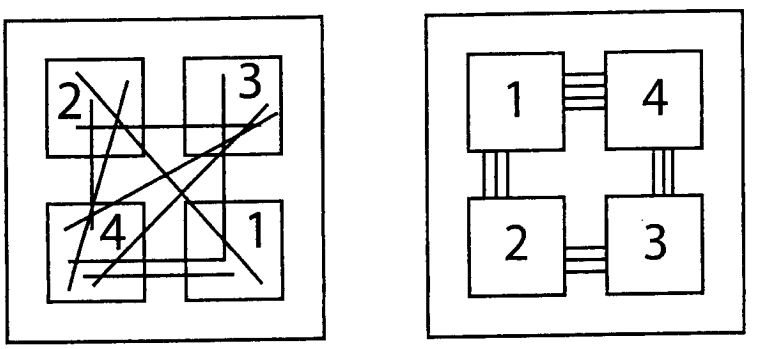


Figure 1–6. *Neat flylines indicate good floorplanning.*

[2] Just as spell checkers knead a human aye.

You will make changes to your block floorplan so that your rat's nest eventually looks as clean, nice, and wireable as possible.[3] You might decide to relocate entire areas of functionality. You might bring one small block across to the other side and fit it between two larger blocks. You might bring a center block to the outside, or an outside block to the center.

When you finally have a block diagram which gives you nice, simple wiring, you save your floorplanning output files.

Timing Checks

Since the final floorplanning tool output files specify where the gates will be generally located, the placement tool roughly knows how long all the wires will be. These wiring length estimations are based on the physical dimensions of the digital library.

Using this information, your floorplanning tool can output an estimated wire length file that goes back into the digital circuit simulator. You now can run some simulations to determine how your estimated wiring lengths will affect your digital circuit. You must check the possibility that long wires will slow the circuit signals too much, affecting the circuit timing.

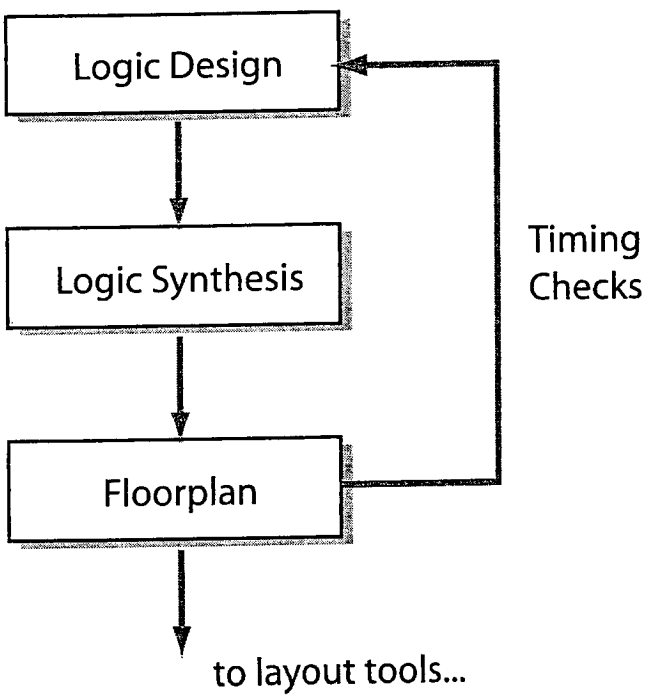


Figure 1–7. *The floorplan/timing loop.*

[3] As Chris always says, "I don't care if it works, as long as it looks good."

If the wire lengths are indeed overly affecting the circuit timing, the designer will need to modify his design, based on your floorplan. He will change the netlist. He might place higher-powered cells in the block to drive the extra wiring capacitances, for example. As the designer works to better organize the design, not only is it easier to wire, but you will find the chip operates better in the end.

You go around this floorplan and timing check loop a couple of times. The simulator will eventually let you know when you have met the timing criteria.

So, at some point, you decide you finally have a good design. You then move on to cementing your devices in place, so to speak. The fine-tuning now begins that we have been putting off.

Placement

We can now nail down the exact positions of all the logic gates within each block, using a placement tool. In the next section, we will then connect the gates with an accurate and final wiring plan, using a routing tool. The placing tool and the routing tool are two of the many software programs that make up the generally named place and route tool suite.

The task of writing one piece of software to do all the placing and routing functions at once would be enormous. Place and route tools are typically broken into individual pieces of software that address specific areas. One piece of software will perform the placement, one piece will perform the wiring, one the block diagramming, and so on.

Some of the software tools have a nice, fancy front end to them that makes it seem as though one program goes off and renders all the output for the chip at once. But, in the background are still many individual programs feeding their outputs to each other, operating one at a time.

There are various differences in the ways place and route tools appear to the user. However, if you pull back the secret curtain, you will see they all essentially work the same way. It is primarily just the user interface that appears slightly different in each of the various software packages sold.

The placement software starts by selecting one block to work with first. It looks for components associated with the selected block. The tool sees 5,000 gates associated with the floating point unit, for example, and wants to place them.

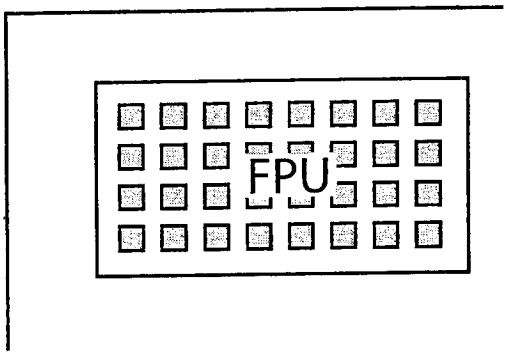


Figure 1–8. Place and route tools place gates in their exact locations.

It might look at the 5,000 gates and basically say, "Well, I see from the floorplan that these 25 gates in the netlist are all to be connected together, so I'll place those gates as close to each other as I can. That way I won't have wires going all over the place." The placement software continues placing logic gates based on their connectivity and the output files from the floorplanning tool.

The initial placement scheme can be considered just a first pass. Depending on the tool, device placement might require multiple passes. The tool says to itself, "Ok, I'll come back and improve the placement, look at it again, improve it a little more, and keep working the placement a little at a time until I get a placement I think is easy to route." The final placement will predetermine much of your eventual layout.

There are placement tools available that can make gate placement decisions based upon the signal timing of a design. This placement approach is known as **timing-driven layout** and has quickly become standard practice. The end result is usually far superior to more traditional techniques.

I/O Drivers

Not only are your gates inside the chip placed at this time, but all the **I/O drivers** are placed at this time, as well. These I/O drivers are the special cells that will drive the input signals, provide outputs, contain ESD protection and test circuitry.

These drivers are placed using separate placement tools that know about the I/O rules. They place the I/O pads separately from the placement of the standard logic cells.

Finally, after all these automated tools and all the timing feedback looping, you have the best placement you think you can reasonably make. Next you will route the wires.

Routing

With your gates and I/O cells nailed in place, you will now start to wire everything together. You will take the file that popped out of the compiler, and drop it into another software program. Again, we rely on automation.[4]

Your wiring software has two priority nets—power and clock signals. It will route these two types of nets first since they are the most critical.

After the power rails and the clock signals have been placed, your wiring software will continue to wire the remainder of the circuitry, beginning with any other circuitry you declare as critical.

We will next examine these specific wiring concerns, in order of importance, beginning with the power nets.

Power Nets

There are certain rules for connecting power to logic gates. Wiring must be centered in certain places and run in certain directions. You end up with power rails running through the middle of your gates, as in Figure 1–9.

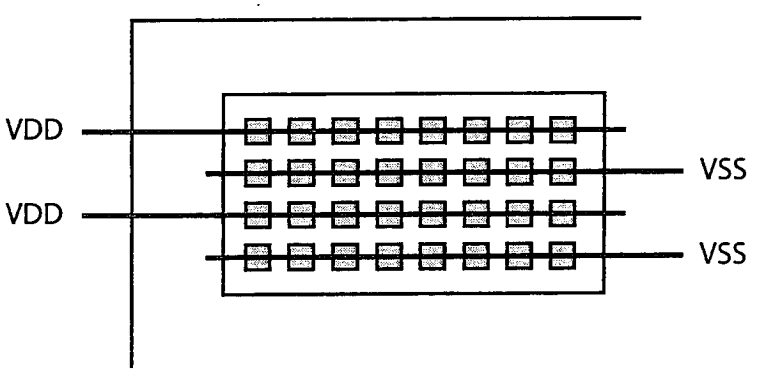


Figure 1–9. *Power rails running through our gates.*

You can intervene in the automated wiring at any time, of course. Perhaps you want to do something special with your power wiring. Maybe you want to move certain blocks around. Maybe you want to put extra power wiring in a particular portion of the circuit because you know the area will demand more power under certain circumstances. It helps if you understand the function of your circuit so that you can make these decisions.

[4] We would not be able to automate so much of our digital layout process without the standardization techniques found in Chapter 2. While these are options available for analog layout, standardization techniques are absolutely required in the digital world. See Chapter 2. (In fact, see all the chapters. After all, you paid for the whole book.)

The wiring software is driven by the net list, which is aware of every component. Therefore, it can tell you when it has completed wiring the power nets.

Strapping

In Figure 1–10, notice the highlighted cell at the far upper right corner, farthest from the VDD input pad. As the power comes into the circuit on the lower left, it must travel through the existing rails. You can see that the supply current has a lot more metal in series to go through to get to our little end cell in the upper right corner.

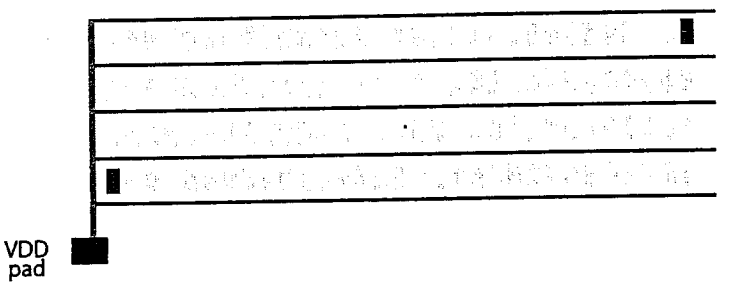


Figure 1–10. *Distant cells see more resistance through the rails due to the length of wire. If only we could lower the resistance to those remote cells.*

The other highlighted cell nearest the pad would see a lot less resistance, being so close. Let's see how we can alleviate this difference.

By laying straps of metal across your power rails, you create a big waffle iron grid of multiple paths. Now it's like having resistors in parallel. The overall resistance reduces.

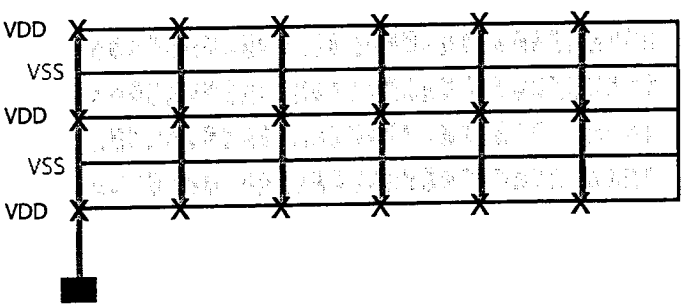


Figure 1–11. *Strapping our rails.*

We have given the current multiple paths. We just blanket the chip with straps and rails to give the chip the most parallel routes for power as possible. The more the merrier.

The older power routers had a uniform spacing rule for power strapping. The power straps were placed every so many microns, regardless of need. You would just say, "Oh, I'll have a metal strap every 60 microns or every 100 microns." Perhaps you thought these were good average numbers, so you would just go with it.

However, newer tools actually look at the drive strengths in the cells before they place their power straps. The router can calculate, for example, that there is a large concentration of huge drive strength cells in certain areas that require more power. So, the router places more power strapping in that area.

Power net wiring is more important than it used to be. More intelligence is being built into the tools to know more about what's going on in the circuits, to give us more intelligent auto-placed and auto-routed chips. It's taking some of the mystery out of the process.

Digital mask designers need to know more about their tools and techniques than they did before we had so much sophistication. They have to understand why the tools are doing what they do. There is more need to understand when the tool messes up, misinterprets, fools itself, or gets stuck. At times we must override the tool's choices. Rules are getting intricate. Tools are getting intricate.

For example, a mask designer might say, "Ok, the power router hasn't done a particularly good job around here. I know this is a very power hungry block, so I'm going to have to go in here and manually put in some extra power strapping."

There can be a lot of manual intervention with digital layout. Using the tools, rebelling against the tools, improving what the tools have routed, finding work the tools left behind, and giving the tools some human touches. People drive the tools.

Clock Net Wiring

Once you have finished running the power rails, your software usually offers a specialized tool just to wire all the clock nets, the circuitry distributing the timing signal. Remember the clock tree synthesis we generated earlier? This is a very critical net.

There are various approaches to wiring a clock net. Each tool has its own particular way of operating.

If you are unfortunate enough to have to hand wire a clock net, you can use the **Central Clock Trunk Approach.** There is usually a clock driver cell that has enough drive strength to drive the top level clock buffers. Place that cell centrally within your design and create a large central trunk that branches out to join to all the clock buffers.

As the net reaches further out from the main driver, it continually splits into more and finer branches. The wire widths at the outer edges become smaller and smaller. The large central region resembles a thick tree trunk, hence the name *Central Clock Trunk Approach.*

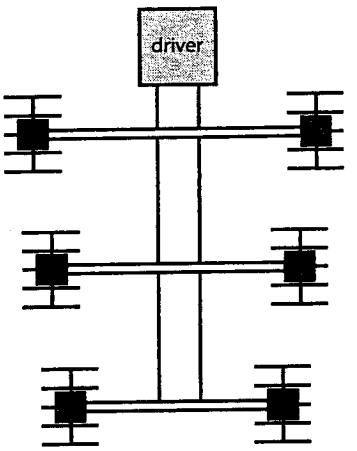


Figure 1–12. Central Clock Trunk Approach to wiring clock nets.

This Central Clock Trunk Approach is very easy to wire. The clock signal distributes very easily.

Other Critical Nets

At this point, we turn to any other nets that need special attention. You wire the nets that you are most worried about first, maybe by hand. You should have been given a list specifying the critical nets of the chip. (If not, ask.)

You plug the critical nets file into the auto-router and it goes off, routing the critical nets while wiring is still relatively easy, before we wire the bulk of the chip. As always, you can intervene at any time until all the critical nets are exactly what you want.

Remaining Nets

The last thing you do is wire the rest of the circuitry. If you are lucky, the tool will know how to automatically wire everything else on its own. Then it says, "I'm done. I'm finished."

Finishing the wiring can take days on really big chips. Typically, the software will go away and work silently on its own. You just sit there and play backgammon. Then just as you discover that elusive guitar chord you've been trying to find, the tool beeps and says, "Finished."[5] Seriously though, there is usually plenty of time to do more productive things while your software is running.

Finishing the Wiring by Hand

When the auto-router finishes as well as it can, you might end up with all the wiring hooked up on your chip. Or, more likely, you might not. Usually, the routers just cannot get to some areas. It can wire itself into some blind alleys and helplessly sit there thinking it is finished. You look at the auto-router's work and ask it, "Ok, how many of the nets couldn't you do?"

Usually you will have to use the human eye, break some nets, and move stuff around, in order to complete the wiring. You can count on some final intervention on your part to finish all the nets and do all the wiring the computer could not do.

For example, the router might say there are five nets it just could not complete. It will highlight for you where these open nets are located. Zoom into the area where the pin is located that it says it cannot wire. Usually in these cases, you will see that the router has placed the wire close by, but it stopped because there is a bunch of stuff in the way. Usually it's something really obvious—a bunch of nets that were pre-wired earlier may be in the way, for example.

Nine times out of ten it's pretty straightforward to move some wires out of the way or do a little re-wiring to free up some area. Then you can normally run the wire where you need to.

It's unlikely the auto-router will do 100% of the wiring in one pass.

With experience, you will often manually pre-wire some signals because you know the auto-router will have problems with them. It saves a lot of time as you learn to preempt these problems in advance.

There are some times, if the chip dimensions are too small, you may not be able to wire the chip at all. You just cannot place 5,000 wires in a tiny 100-micron space.[6] Or, you might have lots of room, but there might be just too many crossed nets, or a bad floorplan. For whatever reason, sometimes you

[5] Chris has a thank you on our band's CD to a certain tool that allowed him the time to practice his guitar licks at work. It's a tool with a reputation for being slow. I'm not sure if they still use that tool, but Chris' guitar licks are still hot and polished.

[6] Yet.

just cannot do it. In that case, start over. Go back to your floorplanning and begin from scratch.

Eventually, you really are finished. All the gates are wired. At this point, you have the computer spit out a wiring file that says, "Ok, here are the real wire lengths, and the real wire capacitances." You have them in hand. No more fine-tuning. No more estimating. This is the real stuff.

Prefabricated Gate Array Chips

All of the above techniques will also be used on a special type of chip, called a **gate array.** A gate array is a predefined and partially prefabricated chip, literally an array of gates. The semiconductor manufacturer processes wafers up to just before the metal is deposited, then stops and stores them until needed.

You will still use floorplanning tools, placement tools, and wiring tools. However, you are not placing any diffusion or poly, only metalization and contact layers. This type of chip is useful for prototyping circuits. Instead of waiting twelve weeks or more for your chip to be fabricated, the processing time for a gate array only takes a few weeks, since you are only fabricating the back end of the process.[7]

Typically, you can choose from only a few fixed-sized gate array die—small, medium, and large. The manufacturer tells you a certain gate array will handle, say, 5,000 gates, this other one will handle 10,000 gates, and this last one will handle 50,000 gates. You select from the offered sizes.

If you have a 30,000-gate design, but your supplier does not have a gate array designed for 30,000 gates, you have to move up to the next size, the 50,000-gate array. You waste a lot of space. Also, the 50,000-gate might have provision for 150 I/O pads, but your design only needs 50 I/O pads. So, with all this extra baggage on the gate array, you end up with a much bigger chip, but you get them built quicker.

When you have finished your logic design, are happy with it, and you have proven your concepts using a gate array, you then convert the design into a real fixed device, a full-custom chip that goes into production.

[7] This reminds me of the old cardpunch days, taking a week to see how my Fortran programs turned out. And I'm talking a week for each attempted fix to the program. Imagine. You kids don't appreciate how we trudged ten miles in the snow, marched thousands of cold cement steps to the dark basement, to drop off a batch job at the one room-sized computer on the UC Davis campus. (Well, it snowed in the California central valley pretty bad in those days.) Now you can make a whole chip in that time.—*Judy*

Verification

It's time to verify that what the team originally had in mind is what was actually built. You will now verify the design.

Design Verification

You feed your new wiring file back to the simulation people again. This time they simulate with the actual wire data. No more estimation. The wiring is physically in place this time.

If you are lucky, everything's happy. If your tools are good enough, and your models are good enough, then the wiring will be fine. If you are not lucky—for instance, the router has done a bad job, or you have done a bad job—the circuit designers may have to change the design again.

If you need to redesign, it is not necessarily a total loss. The team may be able to keep some of the original work—just merge in some new logic, rip out some cells, replace some cells, and re-wire. However, there are times you might have to actually redo the whole enchilada.

The digital mask designer makes these changes. It's a matter of running the tools and software. You typically do not pull out one or two little gates and replace them by hand. The software is intelligent enough to merge in new connectivity and do the majority of the restructuring.

As you can see, a good layout person has to know a lot about the tools and what they are trying to do in the circuit in order to be successful. A good mask designer can put a good floorplan together from the start, especially if they know some details about how the circuit functions. They can preempt problematic issues that they know might develop.

Digital layout is a skill. It is not just a case of pressing the buttons and playing backgammon all day.

You have to know where the tools are going. You have to foresee issues that cause the tools to stumble, and work in advance to prevent those issues. You have to nurse the chip through these processes. You have to direct the flow.

As you master good layout skills, make them part of your daily habit. Soon it all just seems to happen by itself.

Eventually, when all the timing is done, all the wiring is placed, and the chip has been re-simulated, everyone is happy. You have finished the top-level layout of your chip. You have converted a database from a conceptual format into a real mask design.

Physical Verification

Up to this point, you have not been working with real transistors. You have just been working with little boxes with points on them that say, "This is the input. Here is the output. We do not care what is inside." There are no real transistors in those boxes.

GDSII File

To complete your mask design, you take this abstract, high-level database, and replace the boxes with the real logic gates. You merge the real components from real libraries with the wiring and placement data from the place and route tools. As you replace the abstract components with real library components, you produce a **GDSII** stream file of your chip. This is a file that has all the components, all the wiring of your cells, all your vias, everything.

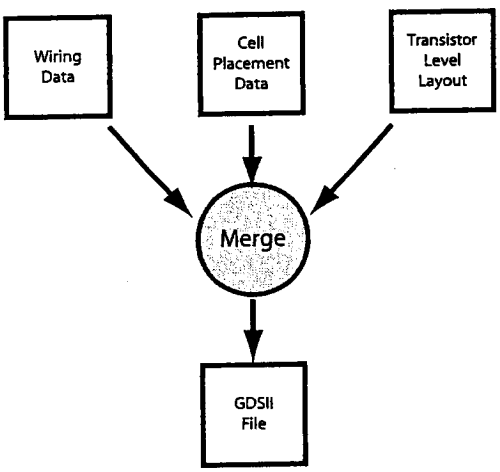


Figure 1–13. The data is merged with the actual transistors to produce a GDSII stream file.

DRC and LVS Checks

By the time you produce your final GDSII output file, the chip has gone through the wringer. Typically, there have been so many operations performed on these databases, from start to finish, that you no longer trust the final output. Has the software kept up with all the loops and changes well enough? Those nice software people do make mistakes, you know, especially when we run lengthy and complex iterations on a project.

Once you get your GDSII stream file, you then will want to run checks to be sure that the wiring is correct and complete. At this point, we check all the process design rules using Design Rule Check (DRC) software.

At the same time, we also check that the wiring and transistor connectivity correctly match the connectivity defined in our netlist. We use Layout Versus Schematic (LVS) software to perform this connectivity check.

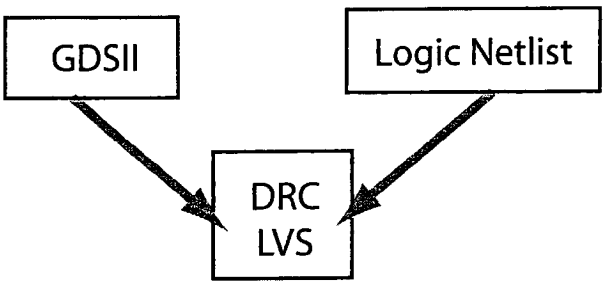


Figure 1–14. *How did we do? Let's run a check.*

Library Management

What could possibly go wrong in this highly automated, computer-controlled nirvana? Plenty.

The place and route process relies on synchronization of all the various database files for all the tools. It relies on the fact that the layout for your inverter is correctly represented by the schematic for the inverter.

Having various representations of the logic gates creates very complicated library management issues. For example, you could have a cell called *Inv1x*, but it might not LVS to the schematic of *Inv1x* because someone has made a mistake in the compiling of the library.

As another example of a possible problem, the place and route tool might not have up-to-date information. The place and route tool might think everything is fine according to what it knows. However, when you merge your cell layout data with the wiring data you could find wires just dangling in free space. The problem could be, for example, that the representation of the inverter the place and route tool uses shows a pin one grid lower than the representation in the GDSII file.

Good library management for digital place and route is very important. An incremental change to one cell in a library could cause six or seven files to require updates in multiple directories. There are so many files, which have

to be 100% perfect and synchronized with each other, mistakes are easily made.

However, if your tools are good, flows are good, and libraries are good, your design should pop out DRC and LVS clean, first time out of the box. If not, you have a bit more work to do. We have a section later in the book to help you resolve these errors. It is not always as difficult as it may first appear.

Summary and Flowchart

So, there you are. That's how large digital chips are created. You have created your netlist. You have placed your gates and routed the wiring. You have considered critical elements of your circuit. Finally, at some point, you are finished.

Tape out. Clean chip. Pass Go. Collect $200.

That was really something, wasn't it?

I feel a flowchart coming on... Yes, let's look at all that again as a flowchart.

The flowchart represents the steps of a typical digital layout process. Follow along in the chart as you read each paragraph below. (See Figure 1-15.)

First you design your logic, synthesize it, draft a floorplan, and then do some timing checks around that loop for a while.

Then you run your place and route tools and do some timing checks. You keep going around that loop until you are happy. You may even have to go around the floorplan loop a couple of times again.

At this point, you need the digital libraries. By library I mean the AND's, OR's, input cells, output cells, and all the real transistor level components. You need the digital libraries to make your final GDSII file.

Finally, you run the DRC and LVS checks against the netlist you started with, out of your logic synthesis. Then you get the final chip done.

You can see from the massive use of computerized tools, that your digital library devices must be put together with a lot of standardization. Your cells, your wiring, and your logic must all be pre-designed with computerized placement in mind.

Analog mask designers get to use standardization techniques as options when appropriate for their projects. However, digital mask design absolutely

Flowchart of Digital Layout Process

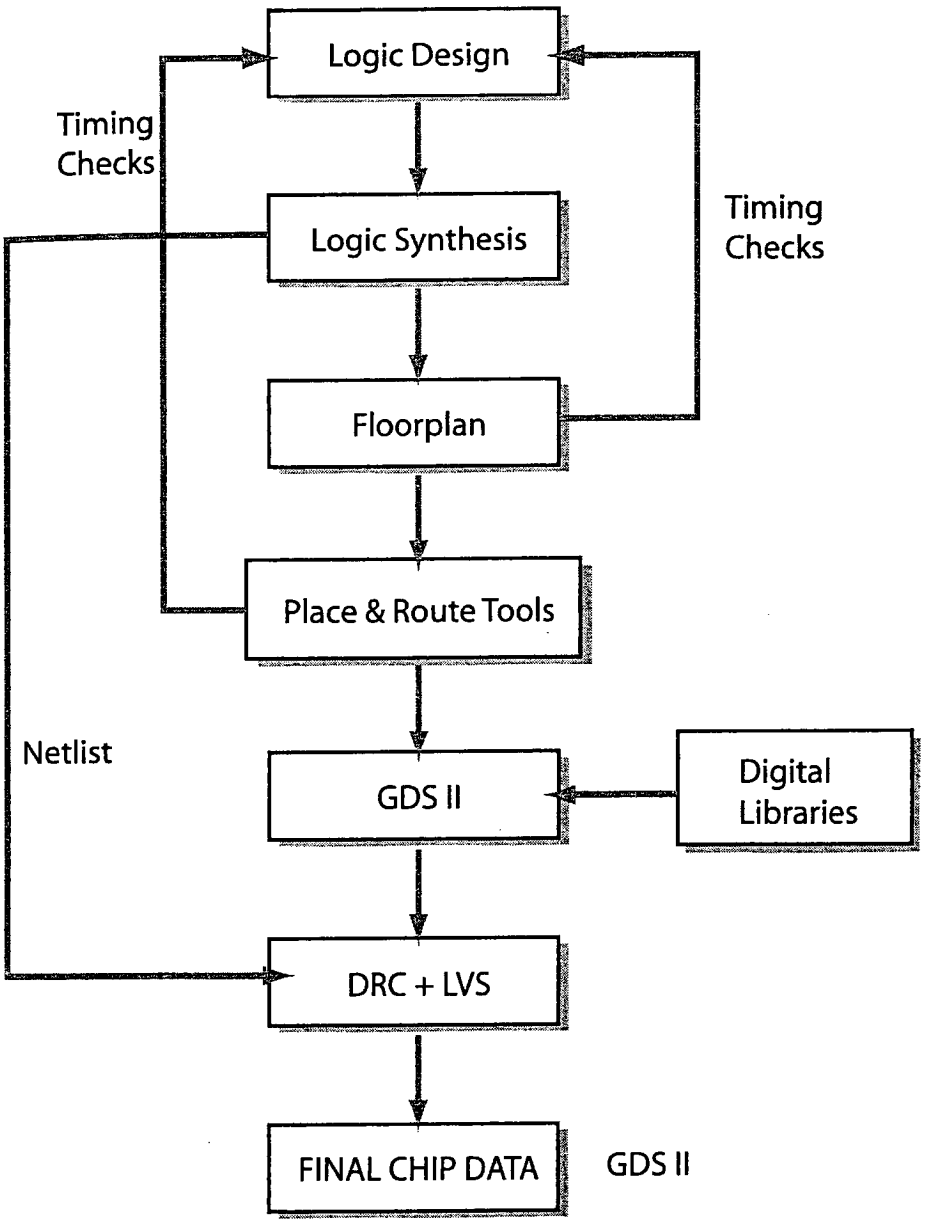


Figure 1–15. *Our original design undergoes many iterative loops and manual interventions before we have our final data for the chip layout.*

demands use of standardization techniques. Automated software tools would not work without it. That's where we go in the next chapter.

Closure on Digital Layout

As you can see, automated digital layout can be very complicated and convoluted. In reality, though, all we are doing is using very simple concepts, just on a massive scale.

Most companies that use these tools to design large digital chips also have very well documented procedures that are designed to take you through every stage of the layout process. There are so many feedback paths and decision points in a large digital design that it is otherwise impossible to keep track of where you are in the flow. You can use the procedures to help you take the project a step at a time.

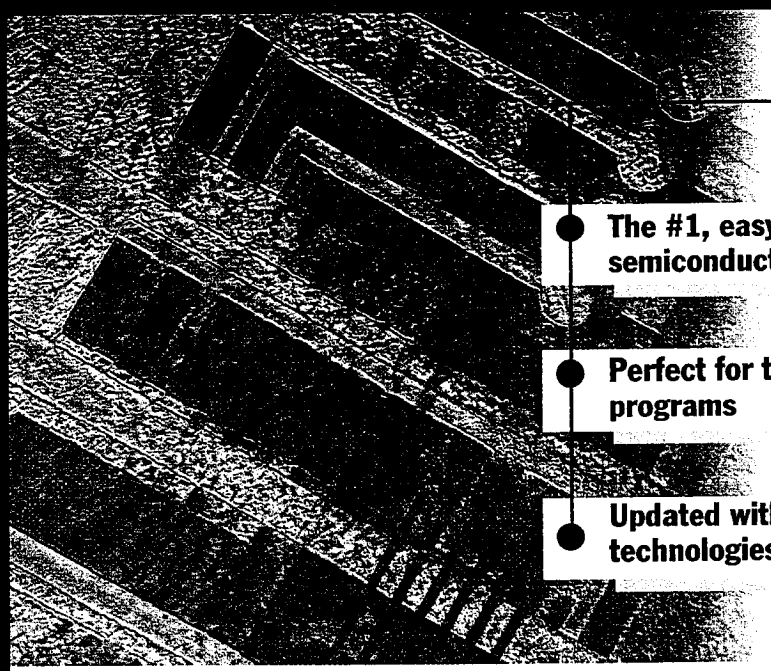
On the face of it, digital layout may appear as though we are just turning a handle and spitting out fully finished chips. However, there is a lot of manual intervention required. The mask designer can stop the process, make adjustments, preset priorities, work ahead of the automation to avoid problems before they happen. The more you know about the automated procedures, the more effective you will be at the helm.

Here's What We've Learned

Here's what you saw in this chapter:

- Place and route tools
- Floorplans and flylines
- Priority nets
- Feedback loops in the design and layout process
- Troubleshooting automated procedures
- Placing buffer cells according to drive strengths
- Gate arrays
- GDSII from place and route tool
- Netlist from logic synthesizer tool
- DRC and LVS checks
- Flowchart of digital layout procedures

The #1 book in the industry — now revised & updated!



- The #1, easy-to-read, math-free introduction to semiconductor processing
- Perfect for training, teaching, & vo-tech programs
- Updated with new cleaning techniques, packing technologies, & fabrication methods

FOURTH EDITION

Microchip Fabrication

A Practical Guide to SEMICONDUCTOR PROCESSING

PETER VAN ZANT

Library of Congress Cataloging-in-Publication Data

Van Zant, Peter.
Microchip fabrication : a practical guide to semiconductor processing / Peter Van Zant.—4th ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-07-135636-3
1. Semiconductors—Design and construction. I. Title.
TK7871.85.V36 2000
621.3815'2—dc21

00-02317

McGraw-Hill
A Division of The McGraw-Hill Companies

4 5 6 7 8 9 0 DOC/DOC 0 9 8 7 6 5 4 3 2

ISBN 0-07-135636-3

The sponsoring editor for this book was Stephen Chapman and the production supervisor was Sherri Souffrance. It was set in Century Schoolbook by Pro-Image Corporation.

Printed and bound by R. R. Donnelley & Sons Company.

This book is printed on recycled, acid-free paper containing a minimum of 50% recycled, de-inked fiber.

each process in turn requires a number of steps and substeps. A speculative process for a 64Gb CMOS device might require 180 major steps, 52 clean/strip, and up to 28 masks.[1] Yet all of the major steps are one of the four basic operations. Figure 4.11 lists the basic operations and the principle process options used for each. In this section, the building of a simple device, an MOS silicon gate transistor, is illustrated to explain a manufacturing sequence. The functions of the individual parts of this type of transistor and the operation of the transistor are explained in Chapter 14.

Circuit design

Circuit design is the first step in creation of a microchip. A circuit designer starts with a block functional diagram of the circuit such as the logic diagram in Fig. 4.12. This diagram lays out the primary functions and operation required of the circuit. Next, the designer translates the functional diagram to a schematic diagram (Fig. 4.13). This diagram identifies the number and connection of the various circuit components. Each component is represented by a symbol. Accompanying the schematic diagram are the electrical parameters (voltage, current, resistance, etc.) required to make the circuit work.

The third step, circuit layout, is unique to semiconductor circuits. Circuit operation is dependent on a number of factors, including material resistivity, material physics, and the physical dimensions of the individual component "parts". Also the placement of the parts relative to each other is another factor. All these considerations dictate the physical layout and dimensions of the part/device/circuit. Layout starts with using sophisticated computer-aided design (CAD) systems to translate each circuit component into the physical shape and size. Through the CAD system, the circuit is built, exactly duplicating the final design. The result is a composite picture of the circuit surface showing all of the sublayer patterns. This drawing is called a *composite* (Fig. 4.14). The composite drawing is analogous to the blueprint of a multistory office building as viewed from the top and showing all of the floors. However, the composite is many times the dimensions of the final circuit.

Both buildings and semiconductor circuits are built one layer at a time. Therefore it is necessary to separate the composite drawing into the layout for each individual layer in the circuit. Figure 4.14 illustrates the composite and individual layer patterns for a simple silicon gate MOS transistor.

Each layer drawing is digitized (digitizing is the translation of the layer drawings to a digital data base) and plotted on a computerized *X-Y* plotting table.

Basic Operation	Process	Options
Layering	Oxidation	Atmospheric
		High Pressure
		Rapid Thermal Oxidation (RTO)
	Chemical Vapor Deposition (CVD)	Atmospheric Pressure
		Low Pressure (LPCVD)
		Plasma Enhanced (PECVD)
		Vapor Phase Epitaxy (VPE)
		Metaloranic CVD (MOCVD)
	Molecular Beam Epitaxy (MBE)	
	Physical Vapor Deposition (PVD)	Vacuum Evaporation
		Sputtering
Patterning	Resist	Positive
		Negative
	Exposure Systems	Contact
		Proximity
		Scanning Projection
		Stepper
	Exposure Sources	High Pressure Mercury
		X-Rays
		E-Beams
	Imaging Processes	Single Layer Resist
		Multilayer Resist
		Antireflecting layers
		Off-Axis Illumination
		Annular Ring Illumination
		Planarization
		Contrast Enhancement
	Etch	Wet Chemistry-Liquid/vapor
		Dry (Plasma)
		Lift-Off
		Ion Milling
		Reactive Ion Etch (RIE)
Doping	Diffusion	Open Tube-Horizontal/Vertical
		Closed Tube
		Rapid Thermal Processing (RTP)
	Ion Implantation	Medium/High Current
		Low/High Voltage (energy)
Heating	Thermal	Hot Plates
		Convection
		RTP
	Radiation	Infrared (IR)

Figure 4.11 Summary of wafer-fab operations/processes.

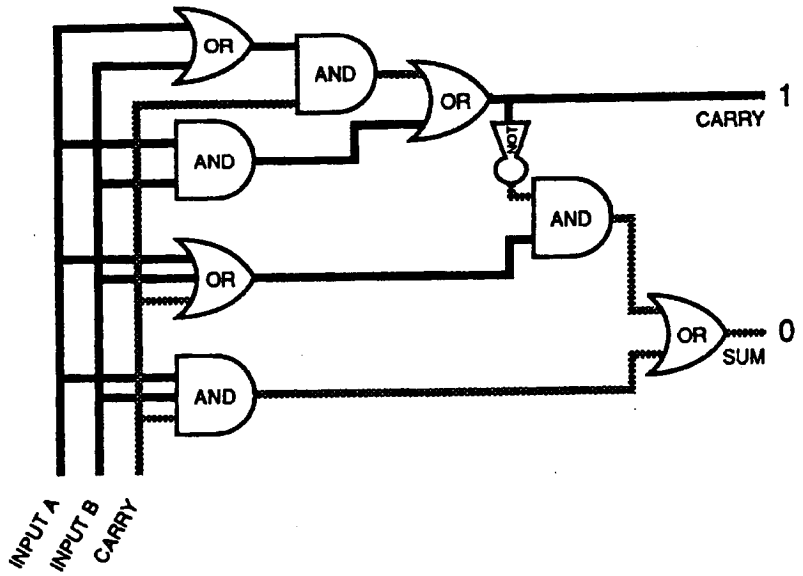


Figure 4.12 Example functional logic design of a simple circuit.

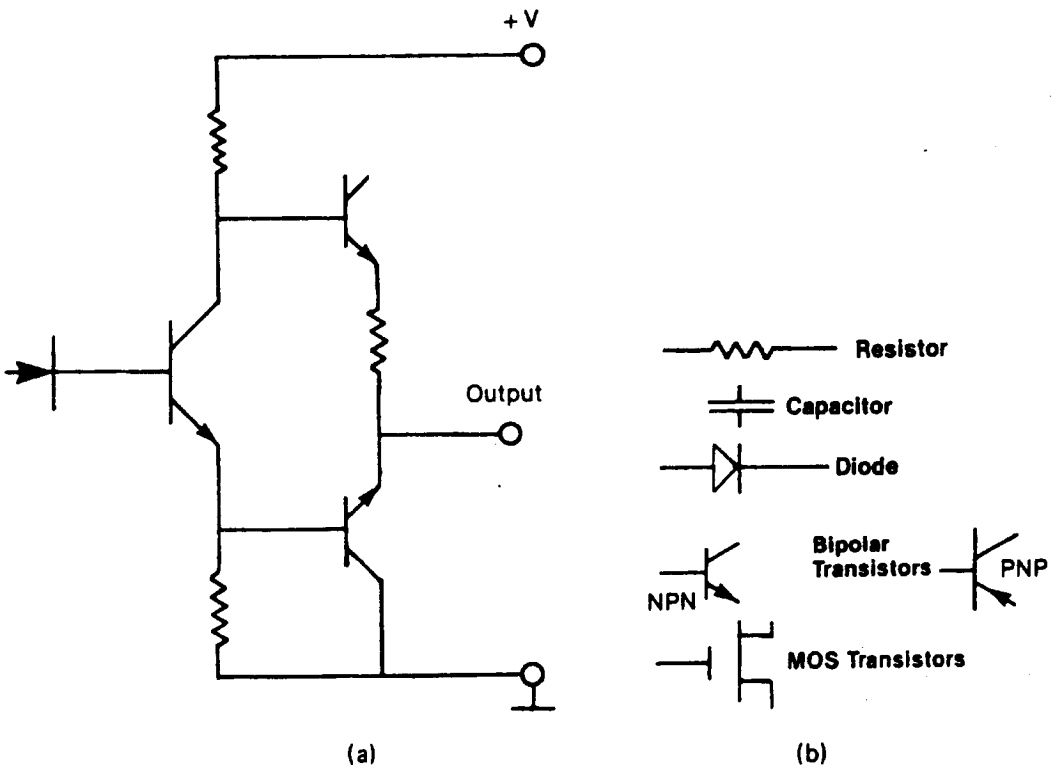


Figure 4.13 Example circuit schematic diagram with component symbols.

Reticle and masks

The patterning process is used to create the required layer pattern and dimensions in and on the wafer surface. Getting the pattern from the digitized pattern to the wafer surface requires several steps. For the photo processes, there is an intermediate step called a reticle. A

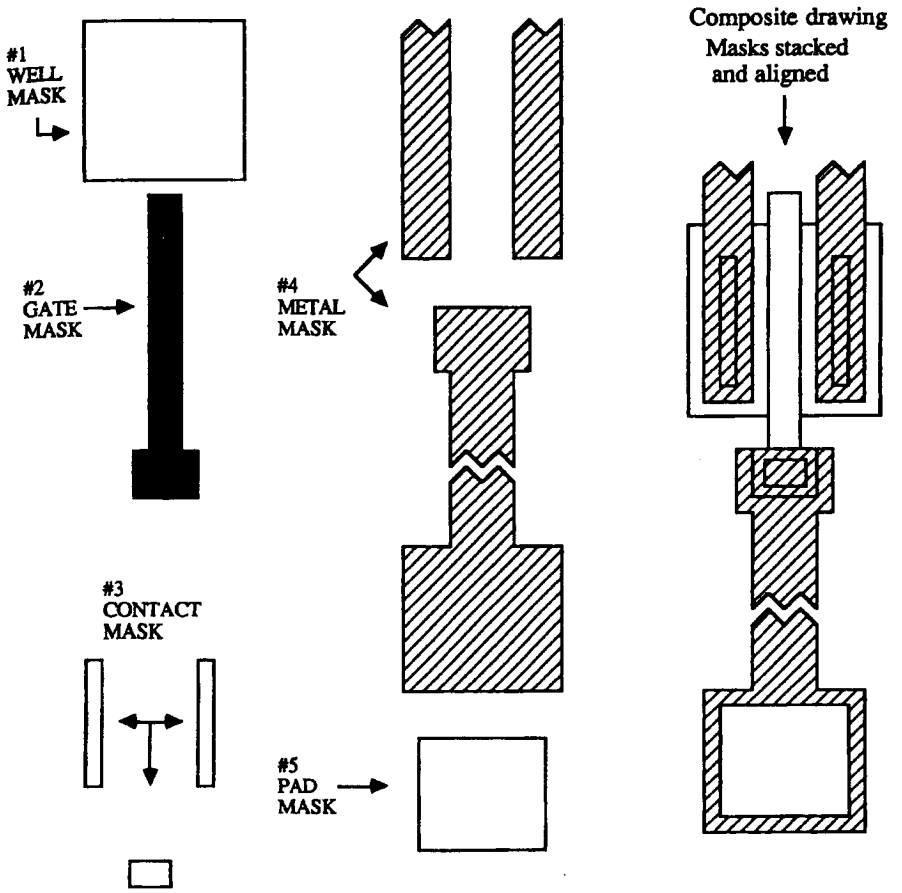


Figure 4.14 Composite and layer drawings for 5-mask silicon gate transistor.

reticle is a "hard copy" of the individual drawing recreated in a thin layer of chrome deposited on a glass or quartz plate (Fig. 4.15*a*). The reticle may be used directly in the patterning process or may be used to produce a photomask. A photomask is also a glass plate with a thin chrome layer on the surface. After production, it is covered with many copies of the circuit pattern (Fig. 4.15*b*). It is used to pattern a whole wafer surface in one pattern transfer. (Reticle and mask-making processes are detailed in Chapter 11.)

The process from circuit design to wafer patterning is shown in Fig. 4.15. Reticles and masks are produced in a separate department or are purchased from outside vendors. They supply the fabrication area with a separate set of reticles or set of masks (*mask set*) for each circuit type.

Example Fabrication Process

The manufacture of a circuit starts with a polished wafer. The cross section sequence in Fig. 4.16 shows the basic operations required to

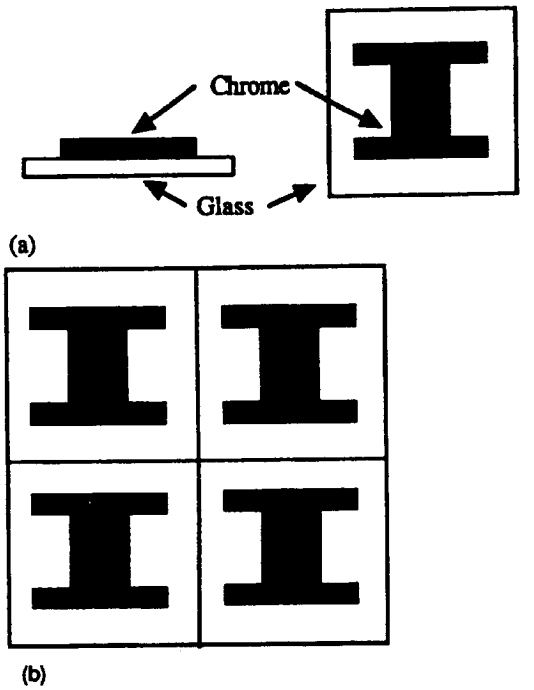


Figure 4.15 (*a*) Chrome on glass reticle. (*b*) Photomask of same pattern.

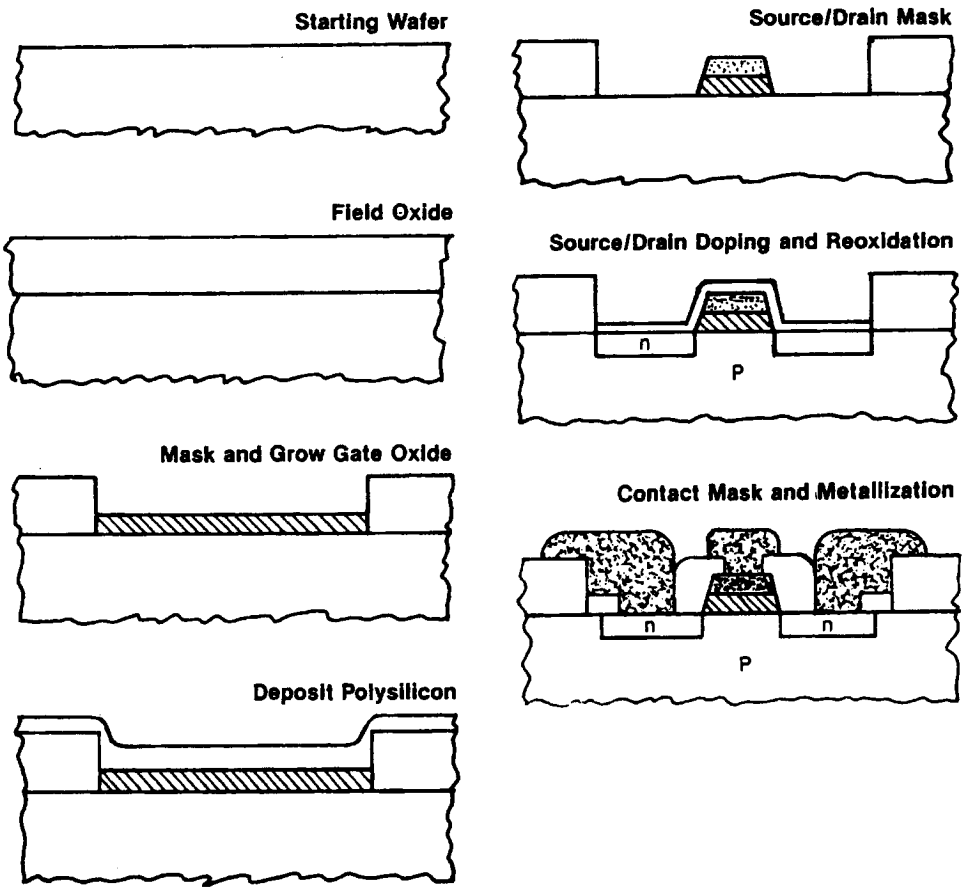


Figure 4.16 Silicon gate MOS process steps.

form a simple MOS silicon-gate transistor structure. Following is an explanation of each operation in the fabrication process.

Step 1: Layering Operation. The building starts with an oxidation of the wafer surface to form a thin protective layer and to serve as a doping barrier. This silicon dioxide layer is called the *field oxide.*

Step 2: Patterning Operation. The patterning process leaves a hole in the field oxide that defines the location of the source, gate, and drain areas of the transistor.

Step 3: Layering Operation. Next, the wafer goes to an silicon dioxide oxidation operation. A thin oxide is grown on the exposed silicon. It will service as the gate oxide.

Step 4: Layering Operation. In step 4, another layering operation is used to deposit a layer of polycrystalline (poly) silicon. This layer will also become part of the gate structure.

Step 5: Patterning Operation. Two openings are patterned in the oxide/polysilicon layer to define the source and drain areas of the transistor.

Step 6: Doping Operation. A doping operation is used to create a n-type pocket in the source and drain areas.

Step 7: Layering Operation. Another oxidation/layering process is used to grow a layer of silicon dioxide over the source/drain areas.

Step 8: Patterning Operation. Holes, called contact holes, are patterned in the source, gate, and drain areas.

Step 9: Layering Operation. A thin layer of conducting metal, usually an aluminum alloy, is deposited over the entire wafer.

Step 10: Patterning Operation. After deposition, the wafer goes back to the patterning area where portions of the metallization layer are removed from the chip area and the scribe lines. The remaining portions connect all the parts of the surface components to each other in the exact pattern required by the circuit design.

Step 11: Heat Treatment Operation. Following the metal patterning step, the wafer goes through a heating process in a nitrogen gas atmosphere. The purpose of the step is to "alloy" the metal to the exposed source and drain regions and the gate region to ensure good electrical contact.

Step 12: Layering Operation. The final layer of this device is a protective layer known variously as a *scratch* or *passivation layer* (not shown in Fig. 4.5). Its purpose is to protect the components on the chip surface during the testing and packaging processes, and during use.

Step 13: Patterning Operation. The last step in the sequence is a patterning process that removes portions of the scratch protection

layer over the metallization terminal pads on the periphery of the chip. This step is known as the *pad mask* (not shown in Fig. 4.6).

The 12-step process illustrates how the four basic fabrication operations are used to build a particular transistor structure. The other components (diodes, resistors, and capacitors) required for the circuit are formed in other areas of the circuit as the tranisitors are being formed. For example, in this sequence resistor patterns are put on the wafer at the same time as the source/drain pattern for the transistor. The subsequent doping operation creates the source/drain *and* the resistors. Other transistor types, such as bipolar and silicon gate MOS, are formed by the same basic four operations, but using different materials and in different sequences.

Chip Terminology

Figure 4.17 is a photomicrograph of an MOS medium-scale integration (MSI) integrated circuit. This level of integration was chosen so that the surface details could be seen. The components of higher-density circuits are so small that they cannot be distinguished on a photomicrograph of the entire chip. The chip features are:

1. A bipolar transistor
2. The circuit designation number
3. Bonding pads for connecting the chip into a package
4. A piece of contamination on a bonding pad
5. Metallization lines
6. Scribe (separation) line
7. Unconnected component
8. Mask alignment marks
9. Resistor

Wafer Sort

Following the wafer-fabrication process comes a very important testing step, wafer sort. This test is the report card on the fabrication process. During the test, each chip is electrically tested for electrical performance and circuit functioning. Wafer sort is also known as *die sort* or *electrical sort*.

dry combinations as the mainstream photoresist removal process. Plasma stripping is used to remove hardened resist layers. A follow-on wet strip step is used to remove the residuals not removed by the plasma.

Post-ion implant and plasma etch stripping

Two problem areas are photresist removal after ion implant and after a plasma strip. Ion implant causes extreme polymerization of the resist and crusting of the top. Generally, the resist is removed or reduced with a dry process, followed by a wet process. Post-plasma etch resist layers are similarily difficult to remove. In addition, the etch process can leave residues, such as $AlCl_3$ and/or $AlBr_3$, that react with water or air forming compounds that corrode the metal system.[34] Low temperature plasma can remove the offending compounds before they take on a corrosive chemistry. Another approach is to add halogens to the plasma atmosphere to minimize the formation of the insoluable metal oxides. This is another instance of setting process parameters to achieve efficient processing (resist removal) without inducing wafer surface damage or metal corrosion.

Final Inspection

The final step in the basic photomasking process is a visual inspection. It is essentially the same procedure as develop inspect, with the exception that the majority of the rejects are fatal (no rework is possible). The one exception is contaminated wafers that may be re-cleaned and reinspected. Final inspection certifies the quality of the outgoing wafers and serves as a check on the effectiveness of the develop inspection. Wafers that should have been identified and pulled from the batch at develop inspect are called *develop inspect escapes*.

The wafers receive a first surface inspection in incident white or ultraviolet light for stains and large particulate contamination. This inspection is followed by a microscopic or automatic inspection for defects and pattern distortions. Measurement of the critical dimensions for the particular mask level is also part of the final inspection. Of primary interest is the quality of the etched pattern with underetching and undercutting being two parameters of concern. The table in Fig. 9.29 is a list of typical causes of wafer rejection found in the final inspection.

Mask Making

In Chapter 5, the steps of circuit design were detailed. In this section, the process used to construct a photomask or reticle is examined. Orig-

Possible Process Cause	Contamination	Misalign	Undercut	Incomplete Etch	Wrong Mask	Pin Holes	C.D.'s	Visual Reject
Contaminated Etch	x		x	x				
Contaminated Stripper	x							
Contaminated H_2O	x							
Insufficient Rinse	x		x					
No Wet Agent				x				
Under Etch				x			x	
Over Etch			x				x	
Wrong Etch			x	x			x	
Hard Bake Too High			x	x			x	
Poor Develop				x			x	
P_2O_5 & SiO_2			x				x	
B_2O_3 & SiO_2				x			x	
Low Hard Bake			x					
Develop Inspect Escapes		x	x	x	x	x		

Figure 9.29 Final inspect rejects and process causes.

inally the masks were made from emulsion-coated glass plates. The emulsions are similar to those found on camera film. These masks were vulnerable to scratches, deteriorated during use, and were not capable of resolving images in the sub 3 micron range. Masks for most modern work use a chrome on glass technology. This mask-making technology is almost identical to the basic wafer-patterning operation (Fig. 9.30). In fact the goal is the same, the creation of a pattern in the thin chrome layer on the glass reticle surface. The preferred materials for mask/reticles are borosilicate glass or quartz, which have good dimensional stability and transmission properties for the wavelengths of the exposing sources. Chrome layers are in the 1000-Å range and deposited on the glass by sputtering (see Chapter 12). Advanced mask/reticles use layers of chromium, chromium oxide, and chromium nitride.[35]

Mask/reticle making follows a number of different paths depending on the starting exposure method (pattern generation, laser, e-beam) and the end result (reticle or mask) (Fig. 9.31). Flow A shows the process for making a reticle using a pattern generator, which is an older technology. A pattern generator consists of a light source and a series of motor-driven shutters. The chrome-covered mask/reticle, with a layer of photoresist is moved under the light source as the shutters are moved and opened to allow precisely shaped patterns of

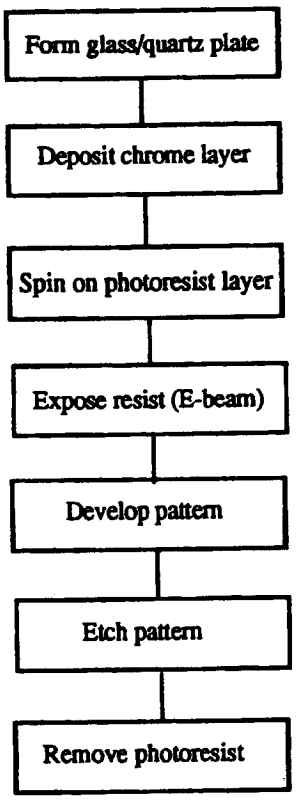


Figure 9.30 Major steps in mask/reticle plate processing.

light to shine onto the resist creating the desired pattern. The reticle pattern is transferred to the resist-covered mask blank by a step-and-repeat process to create a master plate. The master plate is used to create multiple working mask plates in a contact printer. This tool brings the master into contact with a resist-covered mask blank and has a UV light source for transferring the image. After each of the exposure steps (pattern generation, laser, e-beam, master plate expose, and contact print), the reticle/mask is processed through development, inspection, etch, strip, and inspection steps that transfer the pattern permanently into the chrome layer. Inspections are very critical since any undetected mistake or defect has the potential of creating thousands of scrap wafers. Reticles for this use are generally 5 to 20 times the final image size on the mask.[36]

Advanced products with very small geometries and tight alignment budgets require high quality reticle and/or masks. The reticles and masks for these processes are made with lasers or e-beam direct write exposure (Flow A & B). Laser exposure uses a wavelength of 364 nm making it an I-line system. It allows using standard optical resists and is faster than e-beam. Direct write laser sources are turned on and off with an acousto-optical modulator (AOM).[37] In all cases, the reticle or mask is processed to etch the pattern in the chrome.

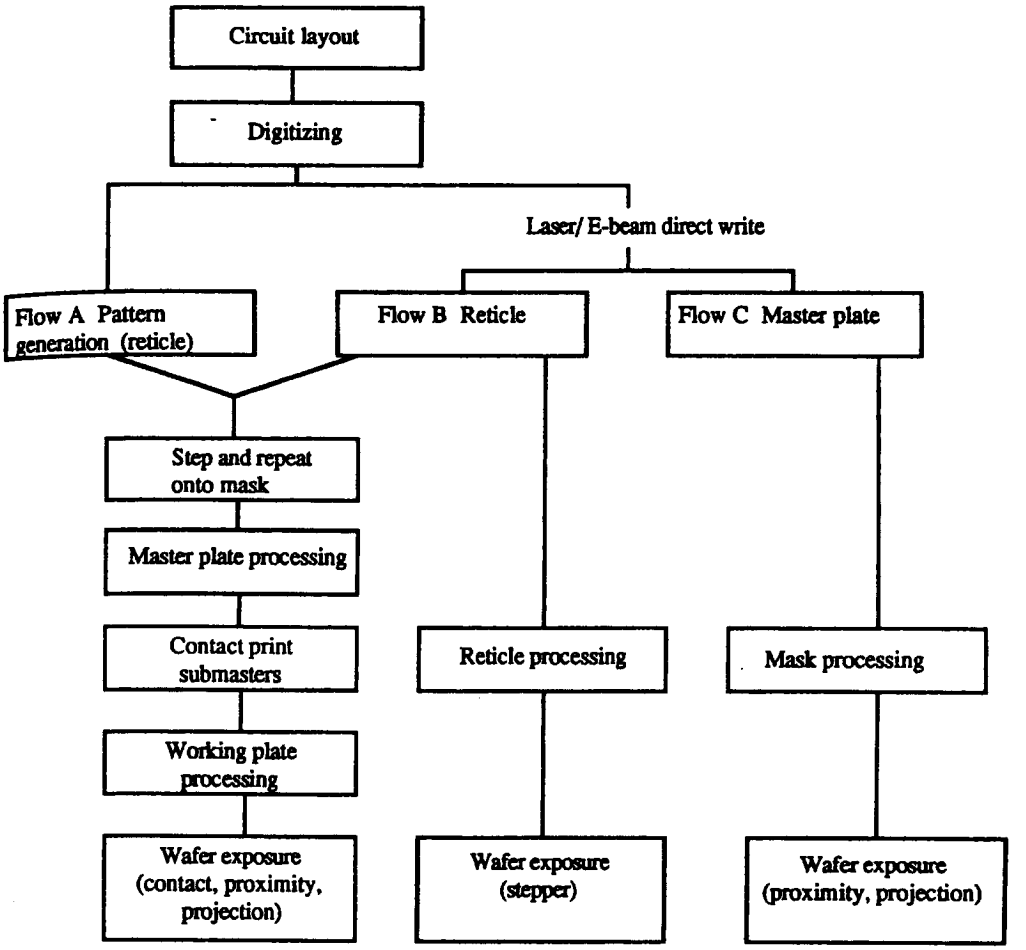


Figure 9.31 Mask/reticle-making processing flows.

Other mask/reticle process flows may be employed. The reticle in Flow A may be laser/e-beam generated or the master plate may be laser/e-beam generated.

VLSI and ULSI-level circuits require virtually defect-free and dimensionally perfect masks and reticles. Critical dimension (CD) budgets from all sources are 10% or better, leaving the reticles with a 4% error margin.[38] There are procedures to eliminate unwanted chrome spots and pattern protrusions with laser "zapping" techniques. Focused ion beams (FIB) is the preferred repair technology for small image masks and reticles. Clear or missing pattern parts are "patched" with a carbon deposit. Opaque or unwanted chrome areas are removed by sputtering from the beam.

Summary

For VLSI and ULSI work, the resolution and registration requirements are very stringent. In 1977, the minimum feature size was 3

μm. By the mid 1980s, it had passed the 1 micron barrier. By the 1990s 0.5 micron sizes were common with 0.35-μm technology planned for production circuits. Circuit design projections call for minimum feature sizes less than 0.1 μm.[39]

Chip manufacturers calculate several *budgets* for each circuit product. A *CD* (critical dimension) *budget* calculates the allowable variation in the image dimensions on the wafer surface. For products with submicron minimum feature sizes, the CD tolerances are 10 to 15%.[40] Also of concern is the critical defect size relative to the minimum feature size. These two parameters are brought together in an *error budget* calculated for the product. Error budgets for various DRAM products are illustrated in Fig. 9.31. An *overlay budget* is the allowable accumulated alignment error for the entire mask set. A rule of thumb is that circuits with micron or submicron feature sizes must meet registration tolerances of one-third the minimum feature. For a 0.35-μm product, the allowable overlay budget is about 0.1 μm.[41]

Key Concepts and Terms

Alignment error budget
Develop inspect and rework
Dry etch methods
Dry stripping
Etch process
Final inspect
Hard bake methods
Hard bake process
Immersion develop
Mask making
Negative resist developers
Plasma descum
Positive resist developers
Puddle develop
Resist development
Resist stripping
Spray develop
Wet etch methods
Wet strip chemicals

Review Questions

1. Name the major methods of resist development.
2. What are the chemicals used to develop negative and positive resist?
3. What is the purpose of the hard bake step?
4. Name three methods used for hard bake.
5. What problems arise if the hard bake temperature is too low? Too high?

AMERICAN NATIONAL STANDARD

ANSI/IEEE Std 100-1988

Fifth Edition
Revised and Expanded

IEEE Standard Dictionary of Electrical and Electronics Terms

PLUS A UNIQUE FEATURE:
OVER 10,000 ACRONYMS DEFINED

The largest current list of acronyms used in electrical and electronics science, business and industry, government, and the military.

ANSI/IEEE Std 100-1988
Fourth Edition

IEEE Standard Dictionary of Electrical and Electronics Terms

Frank Jay
Editor in Chief

J. A. Goetz
Chairman
Standards Coordinating Committee on Definitions (SCC 10)

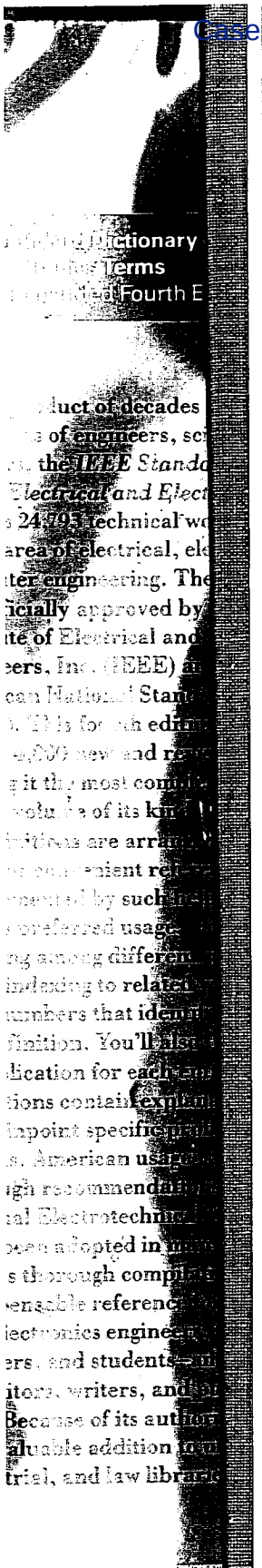
Vice Chairman

ice McClung
d T. Michael*
rd E. Mosher
hn Rankine
S. Robinson
k L. Rose
n M. Wood
H. Zaininger
ald W. Zipse

Membership

Ashcroft, D. L.
Azbill, D. C.
Ball, R. D.
Balaska, T. A.
Bauer, J. T., Jr.
Blasewitz, R. M.
Boberg, R. M.
Boulter, E. A.
Frewin, L. F.
Bucholz, W.
Buckley, F. J.
Cannon, J. B.
Cantrell, R. W.
Chartier, V. L.
Cherney, E. A.
Compton, O. R.
Costrell, L.
Davis, A. M.
Denbrock, F.
DiBlasio, R.
Donnan, R. A.
Duvall, L. M.
Elliott, C. J.
Erickson, C. J.
Flick, C.
Freeman, M.
Gelperin, D.
Guifridda, T. S.
Goldberg, A. A.
Graube, M.
Griffin, C. H.
Heirman, D. N.
Horch, J. W.
James, R. E.
Karady, G. G.
Key, T. S.
Kieburtz, R. B.
Kincaid, M. R.
Klein, R. J.
Klopfenstein, A.
Koepfinger, J. L.
Lensner, W.
Masiello, R. D.
Meitzler, A. H.
Michael, D. T.
Michaels, E. J.
Migliaro, H. W.
Mikulecky, H. W.
Moore, H. R.
Mukhedir, D.
Muller, C. R.
O'Donnell, R. M.
Petersons, O.
Radatz, J.
Reymers, H. E.
Roberts, D. E.
Rosenthal, S. W.
Rothenbukler, W. N.
Sabath, J.
Shea, R. F.
Showers, R. M.
Skomal, E. N.
Smith, T. R.
Smith, E. P.
Smolin, M.
Snyder, J. H.
Spurgin, A. J.
Stephenson, D.
Stepniak, F.
Stewart, R. G.
Swinth, K. L.
Tice, G. D.
Turgel, R. S.
Thomas, L. W., Sr.
Vance, E. E.
Wagner, C. L.
Walter, F. J.
Weinschel, B. O.
Zitovsky, S. A.

Published by
The Institute of Electrical and Electronics Engineers, Inc
New York, NY



Library of Congress Catalog Number 88-082198

ISBN: 1-55937-000-9

November 3, 1988

SH12070

compressor-stator-blade-control system (gas turbines). A means by which the turbine compressor stator blades are adjusted by vary the operating characteristics of the compressor. *See:* **speed-governing system (gas turbines).** 58

computation. *See:* **implicit computation.**

computer (1)(emergency and standby power analog computers). (A) A machine for carrying out calculations. (B) By extension, a machine for carrying out specified transformations on information. 512, 9

(2) (software). (A) A functional unit that can perform substantial computation, including numerous arithmetic operations, or logic operations without intervention by a human operator during a run. (B) A functional programmable unit that consists of one or more associated processing units and peripheral equipment, that is controlled by internally stored programs, and that can perform substantial computation, including numerous arithmetic operations or logic operations, without human intervention. *See:* **programs.** 434

computer-aided design (CAD)(computer applications). The use of computers to aid in design layout and analysis. May included modeling, analysis, simulation, or optimization of designs for production. Often used in combinations such as CAD/CAM. *See:* **computer-aided engineering; computer-aided manufacturing.** 571

computer-aided engineering (CAE)(computer applications). The use of computers to aid in engineering analysis and design. May included solution of mathematical problems, process control, numerical control, and execution of programs performing complex or repetitive calculations. *See:* **computer-aided design; computer-aided manufacturing.** 571

computer-aided inspection (CAI)(computer applications). The use of computers to inspect manufactured parts. 571

computer-aided instruction (CAI)(computer applications). The use of computers to present instructional material and to accept and evaluate student responses. *See:* **computer-based instruction.** 571

computer-aided management (CAM)(computer applications). The application of computers to business management activities. For example, database management, control reporting, and information retrieval. *See:* **decision support system; management information system.** 571

computer-aided manufacturing (CAM)(computer applications). The use of computers and numerical control equipment to aid in manufacturing processes. May include robotics, automation of testing, management functions, control, and product assembly. Often used in combinations such as CAD/CAM. *See:* **computer-aided design; computer-aided engineering.** 571

computer-aided typesetting (computer applications). The use of computers at any stage of the document composition process. This may involve text formatting, input from a word processing system, or computer-aided page makeup. 571

computer-assisted tester (test, measurement and diagnostic equipment). A test not directly programmed by a computer but which operates in association with a computer by using some arithmetic functions of the computer. 54

computer-based instruction (CBI)(computer applications). The use of computers to support any process involving human learning. 571

computer code. A machine code for a specific computer. 255,77

computer component (analog computers). Any part, assembly, or subdivision of a computer, such as resistor, amplifier, power supply, or rack. 9

computer conferencing (computer applications). A form of teleconferencing that allows one or more users to exchange messages on a computer network. 571

computer control (physical process) (electric power systems). A mode of control wherein a computer, using as input the process variables, produces outputs that control the process. *See:* **power system.** 200

computer-control state (analog computers). One of several distinct and selectable conditions of the computer-control circuits. *See:* **balance check; hold; operate; potentiometer set; reset; static test.** 9

computer data (software). Data available for communication between or within computer equipment. Such data can be external (in computer-readable form) or resident within the computer equipment and can be in the form of analog or digital signals. *See:* **computer.** 434

computer diagram (analog computers). A functional drawing showing interconnections between computing elements, such interconnections being specified for the solution of a particular set of equations. *See:* **computer program; problem board.** 9

computer equation (machine equation) (analog computers). An equation derived from a mathematical model for use on a computer which is equivalent or proportional to the original equation. *See:* **scale factor.** 9

computer instruction. A machine instruction for a specific computer. 255, 77

computer-integrated manufacturing (CIM)(computer applications). Use of an integrated system of computer-controlled manufacturing centers. The centers may use robotics, design automation or CAD/CAM (computer-aided design/computer-aided manufacturing technologies. 571

computer interface equipment (surge withstand capability). A device which interconnects a protective relay system to an independent computer, for example, an analog to digital converter, a scanner, a buffer amplifier. 90

computer-managed instruction (CMI)(computer applications). The use of computers for management of student progress. Activities may include record keeping, progress evaluation, and lesson assignment. *See:* **computer-based instruction.** 571

computer network (1) (general). A complex consisting of two or more interconnected computing units. 255, 77

AMERICAN NATIONAL STANDARD

ANSI/IEEE Std 100-1988

I

E

E

E

Standard Dictionary of Electrical and Electronics Terms

The largest current list of acronyms used in electrical and electronics science, business and industry, government, and the military.

ANSI/IEEE Std 100-1988
Fourth Edition

IEEE Standard Dictionary of Electrical and Electronics Terms

Frank Jay
Editor in Chief

J. A. Goetz
Chairman
Standards Coordinating Committee on Definitions (SCC 10)

Membership

Ashcroft, D. L.
Azbill, D. C.
Ball, R. D.
Balaska, T. A.
Bauer, J. T., Jr.
Blasewitz, R. M.
Boberg, R. M.
Boulter, E. A.
Frewin, L. F.
Bucholz, W.
Buckley, F. J.
Cannon, J. B.
Cantrell, R. W.
Chartier, V. L.
Cherney, E. A.
Compton, O. R.
Costrell, L.
Davis, A. M.
Denbrock, F.
DiBlasio, R.
Donnan, R. A.
Duvall, L. M.
Elliott, C. J.
Erickson, C. J.
Flick, C.
Freeman, M.
Gelperin, D.
Guifridda, T. S.
Goldberg, A. A.
Graube, M.
Griffin, C. H.
Heirman, D. N.
Horch, J. W.
James, R. E.
Karady, G. G.
Key, T. S.
Kieburtz, R. B.
Kincaid, M. R.
Klein, R. J.
Klopfenstein, A.
Koepfinger, J. L.
Lensner, W.
Masiello, R. D.
Meitzler, A. H.
Michael, D. T.
Michaels, E. J.
Migliaro, H. W.
Mikulecky, H. W.
Moore, H. R.
Mukhedir, D.
Muller, C. R.
O'Donnell, R. M.
Petersons, O.
Radatz, J.
Reymers, H. E.
Roberts, D. E.
Rosenthal, S. W.
Rothenbukler, W. N.
Sabath, J.
Shea, R. F.
Showers, R. M.
Skomal, E. N.
Smith, T. R.
Smith, E. P.
Smolin, M.
Snyder, J. H.
Spurgin, A. J.
Stephenson, D.
Stepniak, F.
Stewart, R. G.
Swinth, K. L.
Tice, G. D.
Turgel, R. S.
Thomas, L. W., Sr.
Vance, E. E.
Wagner, C. L.
Walter, F. J.
Weinschel, B. O.
Zitovsky, S. A.

Published by
The Institute of Electrical and Electronics Engineers, Inc
New York, NY

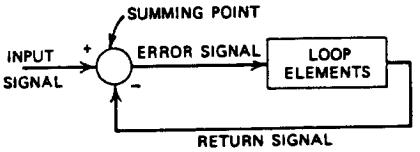
Library of Congress Catalog Number 88-082198

ISBN: 1-55937-000-9

November 3, 1988

SH12070

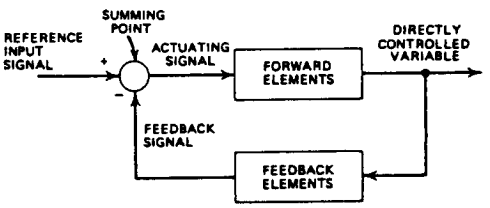
nal resulting from subtracting a particular return signal from its corresponding input signal. See the accompanying figure. *See:* **control system, feedback.** 56, 105



Block diagram of a closed loop

signal, feedback (1) (general). A function of the directly controlled variable in such form as to be used at the summing point. *See:* **control system, feedback.** 206

(2) (control system, feedback). The return signal that results from the reference input signal. See the accompanying figure. *See:* **control system, feedback.** 56, 105



Simplified block diagram indicating essential elements of an automatic control system

signal flow graph (network analysis). A network of directed branches in which each dependent node signal is the algebraic sum of the incoming branch signals at that node. *Note:* Thus,

$$x_1 t_{1k} + x_2 t_{2k} + \ldots + x_n t_{nk} = x_k$$

at each dependent node k, where t_{jk} is the branch transmittance of branch jk. 282

signal frequency shift (frequency-shift facsimile system). The numerical difference between the frequencies corresponding to white signal and black signal at any point in the system. *See:* **facsimile signal (picture signal).** 12

signal generator. A shielded source of voltage or power, the output level and frequency of which are calibrated, and usually variable over a range. *Note:* The output of known waveform is normally subject to one or more forms of calibrated modulation. 185

signal identifier (spectrum analyzer). A means to identify the frequency of the input when spurious responses are possible. A front panel control used to identify the input frequency when spurious responses are present. 390

signal indication. The information conveyed by the aspect of a signal. 328

signaling (1) (data transmission). The production of an audible or visible signal at a station or switchboard by means of an alternating or pulsating current. In a telephone system, any of several methods used to alert subscribers or operators or to establish and control connections. 59

(2) (telephone switching systems). The transmission of address and other switching information between stations and central offices and between switching entities. 55

signaling and doorbell transformers (power and distribution transformer). Step-down transformers (having a secondary of 30 V or less), generally used for the operation of signals, chimes, and doorbells. 53

signaling circuit (National Electrical Code). Any electric circuit that energizes signaling equipment. 256

signaling light. A projector used for directing light signals toward a designated target zone. 167

signal, input (control system, feedback). A signal applied to a system or element. See the figure attached to the definition (3) of **signal, error.** *See:* **control system, feedback.** 56, 105

signal integration. The summation of a succession of signals by writing them at the same location on the storage surface. *See:* **storage tube.** 174

signal level (1)(signals and paths)(microcomputer system bus). The relative magnitude of a signal when compared to an arbitrary reference. Signal levels in ANSI/IEEE Std 796-1983 are specified in volts. 542

(2)(signals and paths)(696 interface devices). The magnitude of a signal when considered in relation to an arbitrary reference magnitude (voltage in the case of ANSI/IEEE Std 696-1983). 538

(3) (programmable instrumentation). The magnitude of signal compared to an arbitrary reference magnitude (voltage in the case of this standard). 378

signal line (programmable instrumentation)(signals and paths)(696 interface devices)(microcomputer system bus). One of a set of signal conductors in an interface system used to transfer messages among interconnected devices. 378, 538, 542

signal lines. The passive transmission lines through which the signal passes from one to another of the elements of the signal transmission system. *See:* **signal.** 188

signalling light (illuminating engineering). A projector used for directing light signals toward a designated target zone. 167

signal off (measuring the performance of tone address signaling systems). Any condition where all the constituent tones of a tone signaling system are below a specified OFF level for each tone. In a single-tone signaling system, tone off and signal off are synonymous terms. *Note:* During a signal off condition, tones that are not used in the signaling system may be at a higher level. 508

A Merriam-Webster®

Webster's Ninth New Collegiate Dictionary

Almost 160,000 entries and 200,000 definitions.

- Entries for words often misused or confused include a clear, authoritative guide to good usage.
- In an exclusive new feature — entries are dated. How old is a word? When was it first used? The answer is here, but in no other American dictionary.
- The newest in the famous Collegiate series, the most widely approved dictionary for home, school and office.

A GENUINE MERRIAM-WEBSTER

The name *Webster* alone is no guarantee of excellence. It is used by a number of publishers and may serve mainly to mislead an unwary buyer.

A Merriam-Webster® is the registered trademark you should look for when you consider the purchase of dictionaries or other fine reference books. It carries the reputation of a company that has been publishing since 1831 and is your assurance of quality and authority.

Library of Congress Cataloging in Publication Data
Main entry under title:

Webster's ninth new collegiate dictionary.

Based on Webster's third new international dictionary.
Includes index.
1. English language—Dictionaries. I. Merriam-Webster Inc.
PE1628.W5638 1987 423 86-23801
ISBN 0-87779-508-8
ISBN 0-87779-509-6 (indexed)
ISBN 0-87779-510-X (deluxe)

Made in the United States of America

2223242526RMc N87

Abbre

as-so-cia-ble \ə-'sō-sh(ē-)ə-bəl, -sē-ə-\ *adj* (1855) : capable of being associated, joined, or connected in thought

¹as-so-ci-ate \ə-'sō-s(h)ē-,āt\ *vb* **-at-ed; -at-ing** [ME *associat* associated, fr. L *associatus*, pp. of *associare* to unite, fr. *ad-* + *sociare* to join, fr. *socius* companion — more at SOCIAL] *vt* (14c) 1 : to join as a partner, friend, or companion 2 *obs* : to keep company with : ATTEND 3 : to join or connect together : COMBINE: *specif* : to subject to chemical association 4 : to bring together or into relationship in any of various intangible ways (as in memory or imagination) ~ *vi* 1 : to come or be together as partners, friends, or companions 2 : to combine or join with other parts : UNITE *syn* see JOIN

²as-so-ciate \ə-'sō-s(h)ē-ət, -shət, -s(h)ē-,āt\ *adj* (14c) 1 : closely connected (as in function or office) with another 2 : closely related esp. in the mind 3 : having secondary or subordinate status ⟨~ membership in a society⟩

³as-so-ciate *like* ²\ *n* (1533) 1 : one associated with another: as a : PARTNER, COLLEAGUE b : COMPANION, COMRADE 2 *often cap* : a degree conferred esp. by a junior college ⟨~ in arts⟩ — **as-so-ciate-ship** \-,ship\ *n*

associate professor *n* (1822) : a member of a college or university faculty who ranks above an assistant professor and below a professor — **associate professorship** *n*

as-so-ci-a-tion \ə-,sō-sē-'ā-shən, -shē-\ *n* (1535) 1 a : the act of associating b : the state of being associated : COMBINATION, RELATIONSHIP 2 : an organization of persons having a common interest : SOCIETY 3 : something linked in memory or imagination with a thing or person 4 : the process of forming mental connections or bonds between sensations, ideas, or memories 5 : the aggregation of chemical species to form (as with hydrogen bonds) loosely bound complexes 6 : a major unit in ecological community organization characterized by essential uniformity and usu. by two or more dominant species — **as-so-ci-a-tion-al** \-shnəl, -shən-'ā\ *adj*

association area *n* (ca. 1909) : an area of the cerebral cortex that functions in linking and coordinating the sensory and motor areas

association football *n* (1873) : SOCCER

as-so-ci-a-tion-ism \ə-,sō-sē-'ā-shə-,niz-əm, -,sō-shē-\ *n* (1875) : a reductionist school of psychology that holds that the content of consciousness can be explained by the association and reassociation of irreducible sensory and perceptual elements — **as-so-ci-a-tion-ist** \-'ā-sh(ə-)nəst\ *n* — **as-so-ci-a-tion-is-tic** \-,ā-shə-'nis-tik\ *adj*

as-so-cia-tive \ə-'sō-s(h)ē-,āt-iv, -shət-iv\ *adj* (1812) 1 : of or relating to association esp. of ideas or images 2 : dependent on or acquired by association or learning 3 : combining elements such that when the order of the elements is preserved the result is independent of the grouping ⟨addition is ~ since $(a + b) + c = a + (b + c)$⟩ — **as-so-cia-tive-ly** *adv* — **as-so-cia-tiv-i-ty** \-,sō-s(h)ē-ə-'tiv-ət-ē, -shə-'tiv-\ *n*

associative learning *n* (1957) : a learning process in which discrete ideas and percepts become linked to one another

associative neuron *n* (1935) : a neuron that conveys impulses from one neuron to another

as-soil \ə-'sȯi(ə)l\ *vt* [ME *assoilen*, fr. OF *assoldre*, fr. L *absolvere* to absolve] (13c) 1 *archaic* : ABSOLVE, PARDON 2 *archaic* : ACQUIT, CLEAR 3 *archaic* : EXPIATE — **as-soil-ment** \-mənt\ *n, archaic*

as-so-nance \'as-ə-nən(t)s\ *n* [F, fr. L *assonare* to answer with the same sound, fr. *ad-* + *sonare* to sound — more at SOUND] (1727) 1 : resemblance of sound in words or syllables 2 a : relatively close juxtaposition of similar sounds esp. of vowels b : repetition of vowels without repetition of consonants (as in *stony* and *holy*) used as an alternative to rhyme in verse — **as-so-nant** \-nənt\ *adj or n*

as soon as *conj* (14c) : immediately at or just after the time that

as-sort \ə-'sȯ(ə)rt\ *vb* [MF *assortir*, fr. *a-* (fr. L *ad-*) + *sorte* sort] *vt* (15c) 1 : to distribute into groups of a like kind : CLASSIFY 2 : to supply with an assortment (as of goods) ~ *vi* 1 : to agree in kind : HARMONIZE 2 : to keep company : ASSOCIATE — **as-sort-er** *n*

as-sor-ta-tive \ə-'sȯrt-ət-iv\ *adj* (1897) : being nonrandom mating based on like or unlike characteristics

as-sort-ed \-'sȯrt-əd\ *adj* (ca. 1797) 1 : suited by nature, character, or design ⟨an ill-*assorted* pair⟩ 2 : consisting of various kinds

as-sort-ment \-'sȯ(ə)rt-mənt\ *n* (ca. 1611) 1 a : the act of assorting b : the state of being assorted 2 : a collection of assorted things or persons

as-suage \ə-'swāj *also* -'swäzh *or* -'swäzh\ *vt* **as-suaged; as-suag-ing** [ME *aswagen*, fr. OF *assouagier*, fr. (assumed) VL *assuaviare*, fr. L *ad-* + *suavis* sweet — more at SWEET] (14c) 1 : to lessen the intensity of (something that pains or distresses) : EASE 2 : PACIFY, QUIET 3 : to put an end to by satisfying : APPEASE, QUENCH ⟨he *assuaged* his hunger with a sandwich⟩ *syn* see RELIEVE — **as-suage-ment** \-mənt\ *n*

as-sua-sive \ə-'swā-siv, -ziv\ *adj* (1708) : SOOTHING, CALMING

as-sume \ə-'süm\ *vt* **as-sumed; as-sum-ing** [ME *assumen*, fr. L *assumere*, fr. *ad-* + *sumere* to take — more at CONSUME] (15c) 1 a : to take up or in : RECEIVE b : to take into partnership, employment, or use 2 a : to take to or upon oneself : UNDERTAKE b : PUT ON, DON 3 : SEIZE, USURP 4 : to pretend to have or be : FEIGN ⟨*assumed* an air of confidence in spite of her dismay⟩ 5 : to take as granted or true : SUPPOSE 6 : to take over (the debts of another) as one's own — **as-sum-abil-i-ty** \-,süm-ə-'bil-ət-ē\ *n* — **as-sum-able** \-'sü-mə-bəl\ *adj* — **as-sum-ably** \-blē\ *adv*

syn ASSUME, AFFECT, PRETEND, SIMULATE, FEIGN, COUNTERFEIT, SHAM mean to put on a false or deceptive appearance. ASSUME often implies a justifiable motive rather than an intent to deceive; AFFECT implies making a false show of possessing, using, or feeling; PRETEND implies an overt and sustained false appearance; SIMULATE suggests a close imitation of the appearance of something; FEIGN implies more artful invention than PRETEND, less specific mimicry than SIMULATE; COUNTERFEIT implies achieving the highest degree of verisimilitude of any of these words; SHAM implies an obvious falseness that fools only the gullible

as-sum-ing *adj* (1695) : PRETENTIOUS, PRESUMPTUOUS

as-sump-sit \ə-'səm(p)-sət\ *n* [NL, he undertook, fr. L *assumere* to undertake] (1590) 1 a : a common-law action alleging damage from a breach of agreement b : an action to recover damages for breach of contract or promise 2 : a promise or contract not under seal on which an action of assumpsit may be brought

as-sump-tion \ə-'səm(p)-shən\ *n* [ME, fr. LL *assumption-, assumptio* taking up, fr. L *assumptus*, pp. of *assumere*] (13c) 1 a : the taking up of a person into heaven b *cap* : August 15 observed in commemoration of the Assumption of the Virgin Mary 2 : a taking to or upon oneself ⟨a delay in the ~ of his new position⟩ 3 : the act of laying claim to or taking possession of something ⟨the ~ of power⟩ 4 : ARROGANCE, PRETENSION 5 a : the supposition that something is true b : a fact or statement (as a proposition, axiom, postulate, or notion) taken for granted 6 : the taking over of another's debts

as-sump-tive \ə-'səm(p)-tiv\ *adj* (1611) 1 : taken as one's own 2 : taken for granted ⟨~ beliefs⟩ 3 : making undue claims : ASSUMING ⟨an ~ person⟩

as-sur-ance \ə-'shůr-ən(t)s\ *n* (14c) 1 : the act or action of assuring: as a : PLEDGE, GUARANTEE b : the act of conveying real property; *also* : the instrument by which it is conveyed c *chiefly Brit* : INSURANCE 2 : the state of being assured: as a : SECURITY b : a being certain in the mind ⟨the puritan's ~ of salvation⟩ c : confidence of mind or manner : easy freedom from self-doubt or uncertainty; *also* : excessive self-confidence : BRASHNESS, PRESUMPTION 3 : something that inspires or tends to inspire confidence ⟨gave repeated ~s of his goodwill⟩ *syn* see CERTAINTY, CONFIDENCE

as-sure \ə-'shů(ə)r\ *vt* **as-sured; as-sur-ing** [ME *assuren*, fr. MF *assurer*, fr. ML *assecurare*, fr. L *ad-* + *securus* secure] (14c) 1 : to make safe (as from risks or against overthrow) : INSURE 2 : to give confidence to : REASSURE 3 : to make sure or certain : CONVINCE 4 : to inform positively ⟨*assured* her of his fidelity⟩ 5 : to make certain the coming or attainment of : GUARANTEE ⟨worked hard to ~ accuracy⟩ *syn* see ENSURE

¹as-sured \ə-'shů(ə)rd\ *adj* (15c) 1 : characterized by certainty or security : GUARANTEED ⟨an ~ market⟩ 2 a : SELF-ASSURED ⟨an ~ dancer⟩ b : SELF-SATISFIED, COMPLACENT 3 : satisfied as to the certainty or truth of a matter : CONVINCED — **as-sured-ly** \-'shůr-əd-lē, -'shů(ə)rd-\ *adv* — **as-sured-ness** \-'shůr-əd-nəs, -'shů(ə)rd-\ *n*

²assured *n, pl* **assured** *or* **assureds** (1755) : INSURED

as-sur-er \ə-'shůr-ər\ *or* **as-sur-or** \ə-'shůr-ər, ə-,shůr-'ȯ(ə)r\ *n* (1607) : one that assures : INSURER

as-sur-gent \ə-'sər-jənt\ *adj* [L *assurgent-, assurgens*, prp. of *assurgere* to rise, fr. *ad-* + *surgere* to rise — more at SURGE] (1578) : moving upward : RISING; *esp* : ASCENDANT 1b

As-syr-i-an \ə-'sir-ē-ən\ *n* (1604) 1 : a member of an ancient Semitic race forming the Assyrian nation 2 : the Semitic language of the Assyrians — **Assyrian** *adj*

As-syr-i-ol-o-gist \ə-,sir-ē-'ä-l-,jəst\ *n* (1865) : a specialist in Assyriology

As-syr-i-ol-o-gy \-jē\ *n* (1828) : the science or study of the history, language, and antiquities of ancient Assyria and Babylonia — **As-syr-i-o-log-i-cal** \-,sir-ē-ə-'lij-i-kəl\ *adj*

-ast \,ast, əst\ *n suffix* [ME, fr. L *-astes*, fr. Gk *-astēs*, fr. verbs in *-azein*] : one connected with ⟨ecdysiast⟩

astar-board \ə-'stär-bərd\ *adv* (1627) : toward or on the starboard side of a ship ⟨put the helm hard ~⟩

As-tar-te \ə-'stärt-ē\ *n* [L, fr. Gk *Astartē*] : the Phoenician goddess of fertility and of sexual love

astat-ic \(')ā-'stat-ik\ *adj* (1827) 1 : not static : not stable or steady 2 : having little or no tendency to take a fixed or definite position or direction — **astat-i-cal-ly** \-i-k(ə-)lē\ *adv* — **astat-i-cism** \-'stat-ə-,siz-əm\ *n*

as-ta-tine \'as-tə-,tēn\ *n* [Gk *astatos* unsteady, fr. *a-* + *statos* standing, fr. *histanai* to cause to stand — more at STAND] (1947) : a radioactive halogen element discovered by bombarding bismuth with helium nuclei and also formed by radioactive decay — see ELEMENT table

as-ter \'as-tər\ *n* (1664) 1 [NL, fr. L, aster, fr. Gk *aster-, astēr* star, aster — more at STAR] a : any of various chiefly fall-blooming leafy-stemmed composite herbs (*Aster* and closely related genera) with often showy heads containing tubular flowers or both tubular and ray flowers b : CHINA ASTER 2 [NL, fr. Gk *aster-, astēr*] : a system of gelated cytoplasmic rays arranged radially about a centrosome at either end of the mitotic or meiotic spindle

-as-ter \,as-tər, 'as-\ *n suffix* [ME, fr. L, suffix denoting partial resemblance] : one that is inferior or not genuine ⟨critic*aster*⟩

as-te-ria \ə-'stir-ē-ə\ *n* [L, a precious stone, fr. Gk, fem. of *asterios* starry, fr. *aster-, astēr*] (1903) : a gemstone cut to show asterism

as-te-ri-at-ed \-ē-,āt-əd\ *adj* [Gk *asterios*] (1816) : exhibiting asterism ⟨~ sapphire⟩

¹as-ter-isk \'as-tə-,risk\ *n* [ME, fr. LL *asteriscus*, fr. Gk *asteriskos*, lit., little star, dim. of *aster-, astēr*] (14c) : the character * used in printing or writing as a reference mark, as an indication of the omission of letters or words, or to denote a hypothetical or unattested linguistic form — **as-ter-isk-less** \-ləs\ *adj*

²asterisk *vt* (ca. 1733) : to mark with an asterisk : STAR

as-ter-ism \'as-tə-,riz-əm\ *n* [Gk *asterismos*, fr. *asterizein* to arrange in constellations, fr. *aster-, astēr*] (1598) 1 a : CONSTELLATION b : a small group of stars 2 : a star-shaped figure exhibited by some crystals by reflected light (as in a star sapphire) or by transmitted light (as in some mica)

astern \ə-'stərn\ *adv or adj* (1627) 1 : behind a ship 2 : at or toward the stern of a ship 3 : STERNFOREMOST, BACKWARD

¹as-ter-oid \'as-tə-,rȯid\ *n* [Gk *asteroeidēs* starlike, fr. *aster-, astēr*] (1802) 1 : one of thousands of small planets between Mars and Jupiter with diameters from a fraction of a mile to nearly 500 miles 2 : STARFISH — **as-ter-oi-dal** \,as-tə-'rȯid-ᵊl\ *adj*

²asteroid *adj* (1854) 1 : resembling a star 2 : of or resembling a starfish

aster yellows *n pl* (1922) : a widespread virus disease that affects more than 40 families of plants, is characterized esp. by yellowing and dwarfing, and is transmitted by leafhoppers

as-the-nia \as-'thē-nē-ə\ *n* [NL, fr. Gk *astheneia*, fr. *asthenēs* weak, fr. *a-* + *sthenos* strength] (1802) : lack or loss of strength : DEBILITY

as-then-ic \as-'then-ik\ *adj* (1789) 1 : of, relating to, or exhibiting asthenia : WEAK 2 : ECTOMORPHIC 2

as-theno-sphere \as-'then-ə-,sfi(ə)r\ *n* [Gk *asthenēs* weak + E *-o-* + *sphere*] (1914) : a hypothetical zone of the earth which lies beneath the lithosphere and within which the material is believed to yield readily to persistent stresses

¹cor·ral \kə-'ral, -'rel\ *n* [Sp, fr. (assumed) VL *currale* enclosure for vehicles, fr. L *currus* cart, fr. *currere* to run — more at CURRENT] (1582) 1 : a pen or enclosure for confining or capturing livestock 2 : an enclosure made with wagons for defense of an encampment

²corral *vt* **cor·ralled; cor·ral·ling** (1847) 1 : to enclose in a corral 2 : to arrange (wagons) so as to form a corral 3 : COLLECT, GATHER ⟨*corralling* votes for the upcoming election⟩

¹cor·rect \kə-'rekt\ *vt* [ME *correcten*, fr. L *correctus*, pp. of *corrigere*, fr. *com-* + *regere* to lead straight — more at RIGHT] (14c) 1 a : to make or set right : AMEND b : COUNTERACT, NEUTRALIZE c : to alter or adjust so as to bring to some standard or required condition ⟨~ a lens for spherical aberration⟩ 2 a : to punish (as a child) with a view to reforming or improving b : to point out usu. for amendment the errors or faults of ⟨spent the day ~*ing* tests⟩ — **cor·rect·able** \-'rek-tə-bəl\ *adj* — **cor·rec·tor** \-'rek-tər\ *n*

syn CORRECT, RECTIFY, EMEND, REMEDY, REDRESS, AMEND, REFORM, REVISE mean to make right what is wrong. CORRECT implies taking action to remove errors, faults, deviations, defects; RECTIFY implies a more essential changing to make something right, just, or properly controlled or directed; EMEND specif. implies correction of a text or manuscript; REMEDY implies removing or making harmless a cause of trouble, harm, or evil; REDRESS implies making compensation or reparation for an unfairness, injustice, or imbalance; AMEND, REFORM, REVISE imply an improving by making corrective changes, AMEND usu. suggesting slight changes, REFORM implying drastic change, and REVISE suggesting a careful examination of something and the making of necessary changes. *syn* see in addition PUNISH

²correct *adj* [ME, corrected, fr. L *correctus*, fr. pp. of *corrigere*] (1676) 1 : conforming to an approved or conventional standard ⟨relations . . . were ~ but not very friendly —W. L. Shirer⟩ 2 : conforming to or agreeing with fact, logic, or known truth 3 : conforming to a set figure ⟨enclosed the ~ return postage⟩ — **cor·rect·ly** \kə-'rek-(t)lē\ *adv* — **cor·rect·ness** \-'rek(t)-nəs\ *n*

syn CORRECT, ACCURATE, EXACT, PRECISE, NICE, RIGHT mean conforming to fact, standard, or truth. CORRECT usu. implies freedom from fault or error; ACCURATE implies fidelity to fact or truth attained by exercise of care; EXACT stresses a very strict agreement with fact, standard, or truth; PRECISE adds to EXACT an emphasis on sharpness of definition or delimitation; NICE stresses great precision and delicacy of adjustment or discrimination; RIGHT is close to CORRECT but has a stronger positive emphasis on conformity to fact or truth rather than mere absence of error or fault.

corrected time *n* (1891) : a boat's elapsed time less her time allowance in yacht racing

cor·rec·tion \kə-'rek-shən\ *n* (14c) 1 : the action or an instance of correcting: as a : AMENDMENT, RECTIFICATION b : REBUKE, PUNISHMENT c : a bringing into conformity with a standard d : NEUTRALIZATION, COUNTERACTION ⟨~ of acidity⟩ 2 : a decline in market price or business activity following and counteracting a rise 3 a : something substituted in place of what is wrong ⟨marking ~*s* on the students' papers⟩ b : a quantity applied by way of correcting (as for adjustment or inaccuracy of an instrument) 4 : the treatment and rehabilitation of offenders through a program involving penal custody, parole, and probation; *also* : the administration of such treatment as a matter of public policy — usu. used in pl. — **cor·rec·tion·al** \-shnəl, -shən-ᵊl\ *adj*

cor·rec·ti·tude \kə-'rek-tə-ˌt(y)üd\ *n* [blend of *correct* and *rectitude*] (1893) : correctness or propriety of conduct

cor·rec·tive \kə-'rek-tiv\ *adj* (1531) : tending to correct ⟨~ lenses⟩ ⟨~ punishment⟩ — **corrective** *n* — **cor·rec·tive·ly** *adv*

¹cor·re·late \'kȯr-ə-ˌlāt, 'kär-, -ˌlət\ *n* [back-formation fr. *correlation*] (1643) 1 : either of two things so related that one directly implies or is complementary to the other (as husband and wife) 2 : a phenomenon (as brain activity) that accompanies another phenomenon (as behavior), is usu. parallel to it (as in form, type, development, or distribution), and is related in some way to it — **correlate** *adj*

²cor·re·late \-ˌlāt\ *vb* **-lat·ed; -lat·ing** *vi* (1742) : to bear reciprocal or mutual relations : CORRESPOND ~ *vt* 1 a : to establish a mutual or reciprocal relation between ⟨~ activities in the lab and the field⟩ b : to show correlation or a causal relationship between 2 : to present or set forth so as to show relationship ⟨he ~*s* the findings of the scientists, the psychologists, and the mystics —Eugene Exman⟩ — **cor·re·lat·able** \-ˌlāt-ə-bəl\ *adj* — **cor·re·lat·or** \-ˌlāt-ər\ *n*

cor·re·la·tion \ˌkȯr-ə-'lā-shən, ˌkär-\ *n* [ML *correlation-*, *correlatio*, fr. L *com-* + *relation-*, *relatio* relation] (1561) 1 : the act of correlating 2 : the state or relation of being correlated; *specif* : a relation existing between phenomena or things or between mathematical or statistical variables which tend to vary, be associated, or occur together in a way not expected on the basis of chance alone ⟨the obviously high positive ~ between scholastic aptitude and college entrance —J. B. Conant⟩ — **cor·re·la·tion·al** \-shnəl, -shən-ᵊl\ *adj*

correlation coefficient *n* (ca. 1909) : a number or function that indicates the degree of correlation between two sets of data or between two random variables and that is equal to their covariance divided by the product of their standard deviations

cor·rel·a·tive \kə-'rel-ət-iv\ *adj* (1530) 1 : naturally related : CORRESPONDING 2 : reciprocally related 3 : regularly used together but typically not adjacent ⟨the ~ conjunctions *either* . . . *or*⟩ — **correlative** *n* — **cor·rel·a·tive·ly** *adv*

cor·re·spond \ˌkȯr-ə-'spänd, ˌkär-\ *vi* [MF or ML; MF *correspondre*, fr. ML *correspondēre*, fr. L *com-* + *respondēre* to respond] (1529) 1 a : to be in conformity or agreement : SUIT ⟨fulfillment seldom ~*s* to anticipation⟩ b : to compare closely : MATCH — usu. used with *to* or *with* c : to be equivalent or parallel 2 : to communicate with a person by exchange of letters ⟨frequently ~*s* with his cousin⟩

cor·re·spon·dence \-'spän-dən(t)s\ *n* (15c) 1 a : the agreement of things with one another b : a particular similarity c : a relation between sets in which each member of one set is associated with one or more members of the other — compare FUNCTION 5a 2 a : communication by letters; *also* : the letters exchanged b : the news, information, or opinion contributed by a correspondent to a newspaper or periodical

correspondence course *n* (1902) : a course offered by a correspondence school

correspondence school *n* (1889) : a school that teaches nonresident students by mailing them lessons and exercises which upon completion are returned to the school for grading

cor·re·spon·den·cy \ˌkȯr-ə-'spän-dən-sē, ˌkär-\ *n, pl* **-cies** (1589) : CORRESPONDENCE

¹cor·re·spon·dent \ˌkȯr-ə-'spän-dənt, ˌkär-\ *adj* [ME, fr. MF or ML; MF, fr. ML *correspondent-*, *correspondens*, prp. of *correspondēre*] (15c) 1 : CORRESPONDING 2 : FITTING, CONFORMING — used with *with* or *to* ⟨the outcome was entirely ~ with my wishes⟩

²correspondent *n* (1630) 1 : something that corresponds 2 a : one who communicates with another by letter b : one who has regular commercial relations with another c : one who contributes news or commentary to a publication (as a newspaper) or a radio or television network often from a distant place ⟨a war ~⟩

cor·re·spond·ing *adj* (1579) 1 a : having or participating in the same relationship (as kind, degree, position, correspondence, or function) esp. with regard to the same or like wholes (as geometric figures or sets) ⟨~ parts of similar triangles⟩ b : RELATED, ACCOMPANYING ⟨all rights carry with them ~ responsibilites —W. P. Paepcke⟩ 2 a : charged with the duty of writing letters ⟨~ secretary⟩ b : participating or serving at a distance and by mail ⟨a ~ member of the society⟩ — **cor·re·spond·ing·ly** \-'spän-diŋ-lē\ *adv*

corresponding angles *n pl* (1797) : any pair of angles each of which is on the same side of one of two lines cut by a transversal and on the same side of the transversal

cor·re·spon·sive \ˌkȯr-ə-'spän(t)-siv, ˌkär-\ *adj* (1606) : mutually responsive

cor·ri·da \kȯ-'rē-thə\ *n* [Sp, lit., act of running] (1896) : BULLFIGHT

cor·ri·dor \'kȯr-əd-ər, 'kär-, -ə-ˌdȯ(ə)r\ *n* [MF, fr. OIt *corridore*, fr. *correre* to run, fr. L *currere* — more at CURRENT] (1814) 1 : a passageway (as in a hotel) into which compartments or rooms open 2 : a usu. narrow passageway or route: as a : a narrow strip of land through foreign-held territory b (1) : a restricted lane for air traffic (2) : a restricted path a spacecraft must follow to accomplish its mission : WINDOW 9 3 : a densely populated strip of land including two or more major cities ⟨the Northeast ~ stretching from Washington into New England —S. D. Browne⟩

cor·rie \'kȯr-ē, 'kär-ē\ *n* [ScGael *coire*, lit., kettle] (1795) : CIRQUE 3

Cor·rie·dale \-ˌdāl\ *n* [*Corriedale*, ranch in New Zealand] (1902) : any of a dual-purpose breed of rather large usu. hornless sheep developed in New Zealand

cor·ri·gen·dum \ˌkȯr-ə-'jen-dəm, ˌkär-\ *n, pl* **-da** \-də\ [L, neut. of *corrigendus*, gerundive of *corrigere* to correct] (1850) : an error in a printed work discovered after printing and shown with its correction on a separate sheet

cor·ri·gi·ble \'kȯr-ə-jə-bəl, 'kär-\ *adj* [ME, fr. MF, fr. ML *corrigibilis*, fr. L *corrigere*] (15c) : capable of being set right : REPARABLE ⟨a ~ defect⟩ — **cor·ri·gi·bil·i·ty** \ˌkȯr-ə-jə-'bil-ət-ē, ˌkär-\ *n*

cor·ri·val \kə-'rī-vəl, kȯ-, kō-\ *n* [MF, fr. L *corrivalis*, fr. *com-* + *rivalis* rival] (1579) : RIVAL, COMPETITOR — **corrival** *adj*

cor·rob·o·rant \kə-'räb-ə-rənt\ *adj, archaic* (1626) : having an invigorating effect — used of a medicine

cor·rob·o·rate \kə-'räb-ə-ˌrāt\ *vt* **-rat·ed; -rat·ing** [L *corroboratus*, pp. of *corroborare*, fr. *com-* + *robor-*, *robur* strength] (1543) : to support with evidence or authority : make more certain *syn* see CONFIRM — **cor·rob·o·ra·tive** \-ˌräb-ə-ˌrā-tiv, -'räb-(ə-)rət-\ *adj* — **cor·rob·o·ra·tor** \-'räb-ə-ˌrāt-ər\ *n* — **cor·rob·o·ra·to·ry** \-'räb-(ə-)rə-ˌtōr-ē, -ˌtȯr-\ *adj*

cor·rob·o·ree \kə-'räb-ə-rē\ *n* [native name in New South Wales, Australia] (1793) 1 : a nocturnal festivity with songs and symbolic dances by which the Australian aborigines celebrate events of importance 2 *Austral* a : a noisy festivity b : TUMULT

cor·rode \kə-'rōd\ *vb* **cor·rod·ed; cor·rod·ing** [ME *corroden*, fr. L *corrodere* to gnaw to pieces, fr. *com-* + *rodere* to gnaw — more at RAT] *vt* (15c) 1 : to eat away by degrees as if by gnawing; *esp* : to wear away gradually usu. by chemical action ⟨the metal was *corroded* beyond repair⟩ 2 : to weaken or destroy gradually : UNDERMINE ⟨manners and miserliness that ~ the human spirit —Bernard DeVoto⟩ ~ *vi* : to undergo corrosion — **cor·rod·ible** \-'rōd-ə-bəl\ *adj*

cor·ro·dy *var of* CORODY

cor·ro·sion \kə-'rō-zhən\ *n* [ME, fr. LL *corrosion-*, *corrosio* act of gnawing, fr. L *corrosus*, pp. of *corrodere*] (15c) 1 : the action, process, or effect of corroding 2 : a product of corroding

cor·ro·sive \-'rō-siv, -ziv\ *adj* (14c) 1 : tending or having the power to corrode ⟨~ acids⟩ ⟨~ action⟩ ⟨the ~ effects of alcoholism⟩ 2 : bitingly sarcastic ⟨~ satire⟩ — **corrosive** *n* — **cor·ro·sive·ly** *adv* — **cor·ro·sive·ness** *n*

corrosive sublimate *n* (1706) : MERCURIC CHLORIDE

cor·ru·gate \'kȯr-ə-ˌgāt, 'kär-\ *vb* **-gat·ed; -gat·ing** [L *corrugatus*, pp. of *corrugare*, fr. *com-* + *ruga* wrinkle — more at ROUGH] *vt* (1620) : to form or shape into wrinkles or folds or into alternating ridges and grooves : FURROW ~ *vi* : to become corrugated

cor·ru·gat·ed *adj* (ca. 1623) : having corrugations ⟨~ paper⟩; *also* : made of corrugated material (as paperboard) ⟨~ boxes⟩

corrugated iron *n* (1856) : usu. galvanized sheet iron or sheet steel shaped into straight parallel regular and equally curved ridges and hollows

cor·ru·ga·tion \ˌkȯr-ə-'gā-shən, ˌkär-\ *n* (1528) 1 : the act of corrugating 2 : a ridge or groove of a surface that has been corrugated

¹cor·rupt \kə-'rəpt\ *vb* [ME *corrupten*, fr. L *corruptus*, pp. of *corrumpere*, fr. *com-* + *rumpere* to break — more at REAVE] *vt* (14c) 1 a : to change from good to bad in morals, manners, or actions; *also* : BRIBE b : to degrade with unsound principles or moral values 2 : ROT, SPOIL 3 : to subject (a person) to corruption of blood 4 : to alter from the original or correct form or version ~ *vi* 1 a : to become tainted or rotten b : to become morally debased 2 : to cause disintegration or

[1]des-cant \'des-ˌkant\ *n* [ME *dyscant*, fr. ONF & ML; ONF *descant*, fr. ML *discantus*, fr. L *dis-* + *cantus* song — more at CHANT] (14c) 1 a : a melody or counterpoint sung above the plainsong of the tenor b : the art of composing or improvising contrapuntal part music; *also* : the music so composed or improvised c : SOPRANO, TREBLE d : a superimposed counterpoint to a simple melody sung typically by some or all of the sopranos 2 : discourse or comment on a theme

[2]des-cant \'des-ˌkant, des-ˈ, dis-ˈ\ *vi* (15c) 1 : to sing or play a descant; *broadly* : SING 2 : COMMENT, DISCOURSE

de-scend \di-ˈsend\ *vb* [ME *descenden*, fr. OF *descendre*, fr. L *descendere*, fr. *de-* + *scandere* to climb — more at SCAN] *vi* (14c) 1 : to pass from a higher place or level to a lower one ⟨~*ed* from the platform⟩ 2 : to pass in discussion from what is logically prior or more comprehensive 3 a : to come down from a stock or source : DERIVE — usu. used in passive ⟨was ~*ed* from an ancient family⟩ b : to pass by inheritance ⟨an heirloom that has ~*ed* in the family⟩ c : to pass by transmission ⟨songs ~*ed* from early ballads⟩ 4 : to incline, lead, or extend downward ⟨the road ~*s* to the river⟩ 5 : to swoop or pounce down or make a sudden attack ⟨the plague ~*ed* upon them⟩ 6 : to proceed in a sequence or gradation from higher to lower or from more remote to nearer or more recent 7 a : to lower oneself in status or dignity : STOOP b : to worsen and sink in condition or estimation ~ *vt* 1 : to pass, move, or climb down or down along 2 : to extend down along — **de-scend-ible** \-ˈsen-də-bəl\ *adj*

[1]de-scen-dant *or* **de-scen-dent** \di-ˈsen-dənt\ *adj* [MF & L; MF *descendant*, fr. L *descendent-*, *descendens*, prp. of *descendere*] (1572) 1 : moving or directed downward 2 : proceeding from an ancestor or source

[2]descendant *or* **descendent** *n* [F & L; F *descendant*, fr. LL *descendent-*, *descendens*, fr. L] (1600) 1 : one descended from another or from a common stock 2 : one deriving directly from a precursor or prototype

de-scend-er \di-ˈsen-dər, ˈdē-ˌ\ *n* (1802) : the part of a lowercase letter (as p) that descends below the main body of the letter; *also* : a letter that has such a part

de-scen-sion \di-ˈsen-chən\ *n, archaic* (14c) : DESCENT 2

de-scent \di-ˈsent\ *n* [ME, fr. MF *descente*, fr. OF *descendre*] (14c) 1 a : derivation from an ancestor : BIRTH, LINEAGE ⟨of French ~⟩ b : transmission or devolution of an estate by inheritance usu. in the descending line c : the fact or process of originating from an ancestral stock d : the shaping or development in nature and character by transmission from a source : DERIVATION 2 : the act or process of descending 3 : a step downward in a scale of gradation; *specif* : one generation in an ancestral line or genealogical scale 4 a : an inclination downward : SLOPE b : a descending way (as a downgrade or stairway) c *obs* : the lowest part 5 a : ATTACK, INVASION b : a sudden disconcerting appearance (as for a visit) 6 : a downward step (as in station or value) : DECLINE ⟨~ of the family to actual poverty⟩

de-scribe \di-ˈskrīb\ *vt* **de-scribed; de-scrib-ing** [L *describere*, fr. *de-* + *scribere* to write — more at SCRIBE] (15c) 1 : to represent or give an account of in words ⟨~ a picture⟩ 2 : to represent by a figure, model, or picture : DELINEATE 3 *obs* : DISTRIBUTE 4 : to trace or traverse the outline of ⟨~ a circle⟩ 5 *archaic* : OBSERVE, PERCEIVE — **de-scrib-able** \-ˈskrī-bə-bəl\ *adj* — **de-scrib-er** *n*

de-scrip-tion \di-ˈskrip-shən\ *n* [ME *descripcioun*, fr. MF & L; MF *description*, fr. L *description-*, *descriptio*, fr. *descriptus*, pp. of *describere*] (14c) 1 a : an act of describing; *specif* : discourse intended to give a mental image of something experienced (as a scene, person, or sensation) b : a descriptive statement or account ⟨a fascinating ~ of his adventures⟩ 2 : kind or character esp. as determined by salient features ⟨opposed to any tax of so radical a ~⟩ *syn* see TYPE

de-scrip-tive \di-ˈskrip-tiv\ *adj* (1751) 1 : serving to describe ⟨a ~ account⟩ 2 : referring to, constituting, or grounded in matters of observation or experience ⟨the ~ basis of science⟩ 3 *of a modifier* a : expressing the quality, kind, or condition of what is denoted by the modified term ⟨*hot* in "hot water" is a ~ adjective⟩ b : NONRESTRICTIVE 4 : of, relating to, or dealing with the structure of a language at a particular time usu. with exclusion of historical and comparative data ⟨~ linguistics⟩ — **de-scrip-tive-ly** *adv* — **de-scrip-tive-ness** *n*

de-scrip-tor \di-ˈskrip-tər\ *n* (1951) : something (as an index term) used to identify an item (as a subject or document) esp. in an information retrieval system

[1]de-scry \di-ˈskrī\ *vt* **de-scried; de-scry-ing** [ME *descrien*, fr. MF *descrier* to proclaim, decry] (14c) 1 a : to catch sight of b : to find out : DISCOVER 2 *obs* : to make known : REVEAL

[2]descry *n, obs* (1605) : discovery or view from afar

Des-de-mo-na \ˌdez-də-ˈmō-nə\ *n* : the wife of Othello in Shakespeare's *Othello*

des-e-crate \ˈdes-i-ˌkrāt\ *vt* **-crat-ed; -crat-ing** [*de-* + *-secrate* (as in *consecrate*)] (1677) 1 : to violate the sanctity of : PROFANE 2 : to treat irreverently or contemptuously often in a way that provokes outrage on the part of others ⟨the kind of shore development . . . that has *desecrated* so many waterfronts — John Fischer⟩ — **des-e-crat-er** *or* **des-e-cra-tor** \-ˌkrāt-ər\ *n*

des-e-cra-tion \ˌdes-i-ˈkrā-shən\ *n* (1717) : an act or instance of desecrating : the state of being desecrated

de-seg-re-gate \(ˌ)dē-ˈseg-ri-ˌgāt\ *vt* (1952) : to eliminate segregation in; *specif* : to free of any law, provision, or practice requiring isolation of the members of a particular race in separate units ~ *vi* : to bring about desegregation

de-seg-re-ga-tion \(ˌ)dē-ˌseg-ri-ˈgā-shən\ *n* (1951) 1 : the act or process or an instance of desegregating 2 : the state of being desegregated

de-se-lect \ˌdē-sə-ˈlekt\ *vt* (1965) : to dismiss (a trainee) from a training program

de-sen-si-tize \(ˌ)dē-ˈsen(t)-sə-ˌtīz\ *vt* (1898) 1 : to make (a sensitized or hypersensitive individual) insensitive or nonreactive to a sensitizing agent 2 : to make (a photographic material) less sensitive or completely insensitive to radiation 3 : to make emotionally insensitive or callous; *specif* : to extinguish an emotional response (as of fear, anxiety, or guilt) to stimuli that formerly induced it — **de-sen-si-ti-za-tion** \(ˌ)dē-ˌsen-sət-ə-ˈzā-shən, -ˌsen-stə-ˈzā-\ *n* — **de-sen-si-tiz-er** \(ˌ)dē-ˈsen-sə-ˌtī-zər\ *n*

[1]des-ert \ˈdez-ərt\ *n* [ME, fr. OF, fr. LL *desertum*, fr. L, neut. of *desertus*, pp. of *deserere* to desert, fr. *de-* + *serere* to join together — more at SERIES] (13c) 1 a : arid barren land; *esp* : a tract incapable of supporting any considerable population without an artificial water supply b : an area of water apparently devoid of life 2 *archaic* : a wild uninhabited and uncultivated tract 3 : a desolate or forbidding area ⟨lost in a ~ of doubt⟩ — **de-ser-tic** \də-ˈzərt-ik\ *adj*

[2]des-ert \ˈdez-ərt\ *adj* (13c) 1 : desolate and sparsely occupied or unoccupied ⟨a ~ island⟩ 2 : of or relating to a desert 3 *archaic* : FORSAKEN

[3]de-sert \di-ˈzərt\ *n* [ME *deserte*, fr. OF, fr. fem. of *desert*, pp. of *deservir* to deserve] (13c) 1 : the quality or fact of deserving reward or punishment 2 : deserved reward or punishment — usu. used in plural ⟨got his just ~*s*⟩ 3 : EXCELLENCE, WORTH

[4]de-sert \di-ˈzərt\ *vb* [F *déserter*, fr. LL *desertare*, fr. *desertus*] *vt* (1603) 1 : to withdraw from or leave usu. without intent to return 2 a : to leave in the lurch ⟨~ a friend in trouble⟩ b : to abandon (military service) without leave ~ *vi* : to quit one's post, allegiance, or service without leave or justification; *esp* : to absent oneself from military duty without leave and without intent to return *syn* see ABANDON — **de-sert-er** *n*

de-sert-i-fi-ca-tion \di-ˌzərt-ə-fə-ˈkā-shən\ *n* [*[1]desert* + *-ification* (as in *saponification*)] (1974) : the process of becoming arid land or desert (as from land mismanagement or climate change)

de-ser-tion \di-ˈzər-shən\ *n* (1591) 1 : an act of deserting; *esp* : the abandonment without consent or legal justification of a person, post, or relationship and the associated duties and obligations ⟨sued for divorce on grounds of ~⟩ 2 : a state of being deserted or forsaken

desert locust *n* (1944) : a destructive migratory locust (*Schistocerca gregaria*) of southwestern Asia and parts of northern Africa

desert soil *n* (ca. 1938) : a soil that develops under sparse shrub vegetation in warm to cool arid climates with a light-colored surface soil usu. underlain by calcareous material and a hardpan layer

de-serve \di-ˈzərv\ *vb* **de-served; de-serv-ing** [ME *deserven*, fr. OF *deservir*, fr. L *deservire* to serve zealously, fr. *de-* + *servire* to serve] *vt* (13c) : to be worthy of : MERIT ⟨~*s* another chance⟩ ~ *vi* : to be worthy, fit, or suitable for some reward or requital ⟨have become recognized as they ~ —T. S. Eliot⟩ — **de-serv-er** *n*

de-served \-ˈzərvd\ *adj* (ca. 1552) : of, relating to, or being that which one deserves ⟨a ~ reputation⟩ — **de-served-ly** \-ˈzər-vəd-lē, -ˈzərv-dlē\ *adv* — **de-served-ness** \-ˈzər-vəd-nəs, -ˈzərv(d)-nəs\ *n*

[1]de-serv-ing \-ˈzər-viŋ\ *n* (14c) : DESERT, MERIT ⟨reward the proud according to their ~*s* —Charles Kingsley⟩

[2]deserving *adj* (1576) : MERITORIOUS, WORTHY; *specif* : meriting financial aid ⟨scholarships for ~ students⟩

de-sex \(ˌ)dē-ˈseks\ *vt* (1911) : CASTRATE, SPAY

de-sex-u-al-ize \(ˌ)dē-ˈseksh-(ə-)wə-ˌlīz, -ˈsek-shə-ˌlīz\ *vt* (1894) 1 : to deprive of sexual characters or power 2 : to divest of sexual quality — **de-sex-u-al-iza-tion** \(ˌ)dē-ˌseksh-(ə-)wə-lə-ˈzā-shən, -ˈsek-shə-lə-\ *n*

des-ha-bille \ˌdes-ə-ˈbē(ə)l, -ˈbil, -ˈbā\ *var of* DISHABILLE

des-ic-cant \ˈdes-i-kənt\ *n* (1676) : a drying agent (as calcium chloride)

des-ic-cate \ˈdes-i-ˌkāt\ *vb* **-cat-ed; -cat-ing** [L *desiccatus*, pp. of *desiccare* to dry up, fr. *de-* + *siccare* to dry, fr. *siccus* dry — more at SACK] *vt* (1575) 1 : to dry up 2 : to preserve (a food) by drying : DEHYDRATE 3 : to drain of emotional or intellectual vitality ~ *vi* : to become dried up — **des-ic-ca-tion** \ˌdes-i-ˈkā-shən\ *n* — **de-sic-ca-tive** \ˈdes-i-ˌkāt-iv, di-ˈsik-ət-\ *adj* — **des-ic-ca-tor** \ˈdes-i-ˌkāt-ər\ *n*

de-sid-er-ate \di-ˈsid-ə-ˌrāt, -ˈzid-\ *vt* **-at-ed; -at-ing** [L *desideratus*, pp. of *desiderare* to desire] (1645) : to entertain or express a wish to have or attain — **de-sid-er-a-tion** \-ˌsid-ə-ˈrā-shən, -ˌzid-\ *n* — **de-sid-er-a-tive** \-ˈsid-ə-ˌrāt-iv, -ˈsid-(ə-)rət-, -ˈzid-\ *adj*

de-sid-er-a-tum \-ˌsid-ə-ˈrāt-əm, -ˌzid-, -ˈrāt-\ *n, pl* **-ta** \-ə\ [L, neut. of *desideratus*] (1652) : something desired as essential

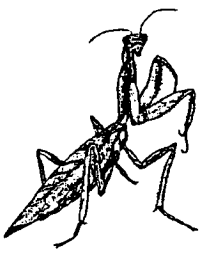
[1]de-sign \di-ˈzīn\ *vb* [MF *designer*, fr. L *designare*, fr. *de-* + *signare* to mark, mark out — more at SIGN] *vt* (1548) 1 a : to conceive and plan out in the mind ⟨he ~*ed* the perfect crime⟩ b : to have as a purpose : INTEND ⟨he ~*ed* to excel in his studies⟩ c : to devise for a specific function or end ⟨a book ~*ed* primarily as a college textbook⟩ 2 *archaic* : to indicate with a distinctive mark, sign, or name 3 a : to make a drawing, pattern, or sketch of b : to draw the plans for c : to create, fashion, execute, or construct according to plan : DEVISE, CONTRIVE ~ *vi* 1 : to conceive or execute a plan 2 : to draw, lay out, or prepare a design — **de-sign-ed-ly** \-ˈzī-nəd-lē\ *adv*

[2]design *n* (1588) 1 a : a particular purpose held in view by an individual or group ⟨he has ambitious ~*s* for his son⟩ b : deliberate purposive planning ⟨battle was joined . . . more by accident than ~ —John Buchan⟩ 2 : a mental project or scheme in which means to an end are laid down 3 a : a deliberate underlying project or scheme : PLOT b *pl* : aggressive or evil intent — used with *on* or *against* ⟨he has ~*s* on the money⟩ 4 : a preliminary sketch or outline showing the main features of something to be executed : DELINEATION 5 a : an underlying scheme that governs functioning, developing, or unfolding : PATTERN, MOTIF ⟨the general ~ of the epic⟩ b : a plan or protocol for carrying out or accomplishing something (esp. a scientific experiment); *also* : the process of preparing this 6 : the arrangement of elements or details in a product or work of art 7 : a decorative pattern 8 : the creative art of executing aesthetic or functional designs *syn* see INTENTION, PLAN

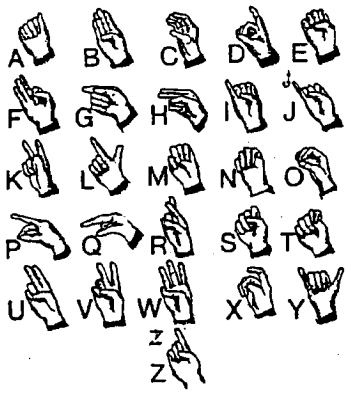
[1]des-ig-nate \ˈdez-ig-ˌnāt, -nət\ *adj* [L *designatus*, pp. of *de[illegible]are*] (1646) : chosen for an office but not yet installed ⟨ambassador

[2]des-ig-nate \-ˌnāt\ *vt* **-nat-ed; -nat-ing** (1791) 1 : to [illegible]t apart for a specific purpose, office, or duty 2 a : location of ⟨a marker *designating* the crest of the flo[illegible] DICATE ⟨any task *designated* by the employer⟩ c : class ⟨the area we ~ as that of spiritual values : SPECIFY, STIPULATE 3 : DENOTE 4 : to call by a

manse \'man(t)s\ *n* [ME *manss*, fr. ML *mansa, mansus, mansum*, fr. L *mansus*, pp. of *manēre*] (15c) 1 *archaic* : the dwelling of a householder 2 : the residence of a clergyman; *esp* : the house of a Presbyterian clergyman 3 : a large imposing residence
man-ser-vant \'man-ˌsər-vənt\ *n, pl* **men-ser-vants** \'men-ˌsər-vən(t)s\ (14c) : a male servant
-man-ship \-mən-ˌship\ *n suffix* [*sportsmanship*] : the art or practice of maneuvering to gain a tactical advantage ⟨gamesmanship⟩
man-sion \'man-chən\ *n* [ME, fr. MF, fr. L *mansion-, mansio*, fr. *mansus*, pp. of *manēre* to remain, dwell; akin to Gk *menein* to remain] (14c) 1 **a** *obs* : the act of remaining or dwelling : STAY **b** *archaic* : DWELLING, ABODE 2 **a** (1) : the house of the lord of a manor (2) : a large imposing residence **b** : a separate apartment or lodging in a large structure 3 **a** : HOUSE 3b **b** : one of the 28 parts into which the moon's monthly course through the heavens is divided
man-size \'man-ˌsīz\ *or* **man-sized** \-ˌsīzd\ *adj* (1913) 1 : suitable for or requiring a man ⟨a ~ job⟩ 2 : larger than others of its kind ⟨constructed a ~ model⟩
man-slaugh-ter \'man-ˌslȯt-ər\ *n* (15c) : the unlawful killing of a human being without express or implied malice
man-slay-er \-ˌslā-ər\ *n* (14c) : one who slays a man
man-sue-tude \'man(t)-swi-ˌt(y)üd, man-'sü-ə-\ *n* [ME, fr. L *mansuetudo*, fr. *mansuetus* tame, mild, fr. pp. of *mansuescere* to tame, fr. *manus* hand + *suescere* to accustom; akin to Gk *ēthos* custom — more at MANUAL, ETHICAL] (14c) : the quality or state of being gentle : MEEKNESS, TAMENESS
man-ta \'mant-ə\ *n* [Sp, alter. of *manto* cloak, fr. L *mantus* short cloak; prob. akin to L *mantellum* mantle] (1697) 1 : a square piece of cloth or blanket used in southwestern U.S. and Latin America usu. as a cloak or shawl 2 [AmerSp, fr. Sp; fr. its shape] : DEVILFISH 1
man-tai-lored \'man-ˌtā-lərd\ *adj* (1922) : made with the severe simplicity associated with men's coats and suits
manta ray *n* (1936) : DEVILFISH 1
man-teau \man-'tō, 'man-ˌ\ *n* [F, fr. OF *mantel*] (1671) : a loose cloak, coat, or robe
man-tel \'mant-ᵊl\ *n* [ME, fr. MF, fr. OF, mantle] (15c) 1 **a** : a beam, stone, or arch serving as a lintel to support the masonry above a fireplace **b** : the finish around a fireplace 2 : a shelf above a fireplace
man-telet \'mant-lət, -ᵊl-ət, ˌmant-ᵊl-'et\ *n* [ME, fr. MF *mantlet*, dim. of *mantel*] (14c) 1 : a very short cape or cloak 2 *or* **mant-let** \'mant-lət\ : a movable shelter formerly used by besiegers as a protection when attacking
man-tel-piece \'mant-ᵊl-ˌpēs\ *n* (1686) 1 : a mantel with its side elements 2 : MANTEL 2
man-tel-shelf \-ˌshelf\ *n* (ca. 1828) : MANTEL 2
man-tic \'mant-ik\ *adj* [Gk *mantikos*, fr. *mantis*] (1850) : of or relating to the faculty of divination : PROPHETIC
man-ti-core \'mant-i-ˌkō(ə)r, -ˌkȯ(ə)r\ *n* [ME, fr. L *mantichora*, fr. Gk *mantichōras*] (14c) : a legendary animal with the head of a man, the body of a lion, and the tail of a dragon or scorpion
man-tid \'mant-əd\ *n* [NL *Mantidae*, group name, fr. *Mantis*, genus name] (1895) : MANTIS
man-til-la \man-'tē-(y)ə, -'til-ə\ *n* [Sp, dim. of *manta*] (1717) 1 : a light scarf worn over the head and shoulders esp. by Spanish and Latin American women 2 : a short light cape or cloak
man-tis \'mant-əs\ *n, pl* **man-tis-es** *or* **man-tes** \'man-ˌtēz\ [NL, fr. Gk, lit., diviner, prophet; akin to Gk *mainesthai* to be mad — more at MANIA] (1658) : an insect (order Manteodea and esp. genus *Mantis*) that feeds on other insects and clasps its prey in forelimbs held up as if in prayer


mantis

man-tis-sa \man-'tis-ə\ *n* [L *mantisa, mantisso* makeweight, fr. Etruscan] (ca. 1847) : the decimal part of a logarithm
¹man-tle \'mant-ᵊl\ *n* [ME *mantel*, fr. OF, fr. L *mantellum*] (13c) 1 **a** : a loose sleeveless garment worn over other clothes : CLOAK **b** : a mantle regarded as a symbol of preeminence or authority ⟨invested his people with the ~ of universal champions of justice —Denis Goulet⟩ 2 **a** : something that covers, enfolds, or envelops **b** (1) : a fold or lobe or pair of lobes of the body wall of a mollusk or brachiopod that lines the shell in shell-bearing forms and bears shell-secreting glands (2) : the soft external body wall that lines the test or shell of a tunicate or barnacle **c** : the outer wall and casing of a blast furnace above the hearth; *broadly* : an insulated support or casing in which something is heated 3 : the back, scapulars, and wings of a bird 4 : a lacy hood or sheath of some refractory material that gives light by incandescence when placed over a flame 5 **a** : MANTLEROCK **b** : the part of the interior of a terrestrial planet and esp. the earth that lies beneath the lithosphere and above the central core 6 : MANTEL
²mantle *vb* **man-tled; man-tling** \'mant-liŋ, -ᵊl-iŋ\ *vt* (13c) : to cover with or as if with a mantle : CLOAK ~ *vi* 1 : to become covered with a coating 2 : to spread over a surface 3 : BLUSH ⟨her rich face *mantling* with emotion —Benjamin Disraeli⟩
man-tle-rock \'mant-ᵊl-ˌräk\ *n* (1895) : unconsolidated residual or transported material that overlies the earth's solid rock
man-to-man \ˌman-tə-'man\ *adj* (1925) 1 : characterized by frankness and honesty ⟨a ~ talk⟩ 2 : of, relating to, or being a system of defense (as in football or basketball) in which each defensive player guards a specified opponent
Man-toux test \ˌman-ˌtü-, ˌmä̃-\ *n* [Charles *Mantoux* †1947 Fr. physician] (ca. 1923) : an intracutaneous test for hypersensitivity to tuberculin that indicates past or present infection with tubercle bacilli
man-tra \'man-trə, 'mən-\ *n* [Skt, sacred counsel, formula, fr. *manyate* he thinks; akin to L *mens* mind — more at MIND] (1808) : a mystical formula of invocation or incantation (as in Hinduism) — **man-tric** \-trik\ *adj*
man-trap \'man-ˌtrap\ *n* (1788) : a trap for catching men : SNARE
man-tua \'manch-(ə-)wə, 'mant-ə-wə\ *n* [modif. of F *manteau* mantle] (1678) : a usu. loose-fitting gown worn esp. in the 17th and 18th centuries
Manu \'man-(ˌ)ü\ *n* [Skt] : the progenitor of the human race and giver of the religious laws of Manu according to Hindu mythology
¹man-u-al \'man-yə(-wə)l\ *adj* [ME *manuel*, fr. MF, fr. L *manualis*, fr. *manus* hand; akin to OE *mund* hand, Gk *marē*] (15c) 1 **a** : of, relating to, or involving the hands (~ dexterity) **b** : worked or done by hand and not by machine ⟨a ~ choke⟩ ⟨~ computation⟩ ⟨~ indexing⟩ 2 : requiring or using physical skill and energy ⟨~ labor⟩ ⟨~ workers⟩ — **man-u-al-ly** \-ē\ *adv*
²manual *n* (15c) 1 : a book that is conveniently handled; *esp* : HANDBOOK 2 : the prescribed movements in the handling of a weapon or other military item during a drill or ceremony ⟨the ~ of arms⟩ 3 **a** : a keyboard for the hands; *specif* : one of the several keyboards of an organ or harpsichord that controls a separate division of the instrument **b** : a device or apparatus intended for manual operation
manual alphabet *n* (ca. 1864) : an alphabet for deaf-mutes in which the letters are represented by finger positions



manual alphabet

manual training *n* (1880) : a course of training to develop skill in using the hands and to teach practical arts (as woodworking and metalworking)
ma-nu-bri-um \mə-'n(y)ü-brē-əm\ *n, pl* **-bria** \-brē-ə\ *also* **-bri-ums** [NL, fr. L, handle, fr. *manus*] (ca. 1848) : an anatomical process or part shaped like a handle: as **a** : the cephalic segment of the sternum of man and many other mammals **b** : the process that bears the mouth of a hydrozoan : HYPOSTOME
man-u-fac-to-ry \ˌman-(y)ə-'fak-t(ə-)rē\ *n* (1647) : FACTORY 2a
¹man-u-fac-ture \ˌman-(y)ə-'fak-chər\ *n* [MF, fr. L *manu factus*, lit., made by hand] (1567) 1 : something made from raw materials by hand or by machinery 2 **a** : the process of making wares by hand or by machinery esp. when carried on systematically with division of labor **b** : a productive industry using mechanical power and machinery 3 : the act or process of producing something
²manufacture *vb* **-tured; -tur-ing** \-'fak-chə-riŋ, -'fak-shriŋ\ *vt* (1683) 1 : to make into a product suitable for use 2 **a** : to make from raw materials by hand or by machinery **b** : to produce according to an organized plan and with division of labor 3 : INVENT, FABRICATE 4 : to produce as if by manufacturing : CREATE ⟨writers who ~ stories for television⟩ ~ *vi* : to engage in manufacture *syn* see MAKE — **man-ufacturing** *n*
man-u-fac-tur-er \ˌman-(y)ə-'fak-chər-ər, -'fak-shrər\ *n* (1719) : one that manufactures; *esp* : an employer of workers in manufacturing
man-u-mis-sion \ˌman-yə-'mish-ən\ *n* [ME, fr. MF, fr. L *manumission-, manumissio*, fr. *manumissus*, pp. of *manumittere*] (15c) : the act or process of manumitting; *esp* : formal emancipation from slavery
man-u-mit \ˌman-yə-'mit\ *vt* **-mit-ted; -mit-ting** [ME *manumitten*, fr. MF *manumitter*, fr. L *manumittere*, fr. *manus* hand + *mittere* to let go, send] (15c) : to release from slavery *syn* see FREE
¹ma-nure \mə-'n(y)u̇(ə)r\ *vt* **ma-nured; ma-nur-ing** [ME *manouren*, fr. MF *manouvrer*, lit., to do work by hand, fr. L *manu operare* — more at MANEUVER] (15c) 1 *obs* : CULTIVATE 2 : to enrich (land) by the application of manure — **ma-nur-er** *n*
²manure *n* (1549) : material that fertilizes land; *esp* : refuse of stables and barnyards consisting of livestock excreta with or without litter — **ma-nu-ri-al** \-'n(y)u̇r-ē-əl\ *adj*
ma-nus \'mā-nəs, 'mä-\ *n, pl* **ma-nus** \-nəs, -ˌnüs\ [NL, fr. L, hand] (ca. 1823) : the distal segment of the vertebrate forelimb including the carpus and forefoot or hand
¹manu-script \'man-yəˌskript\ *adj* [L *manu scriptus*] (1597) : written by hand or typed ⟨~ letters⟩
²manuscript *n* (1600) 1 : a written or typewritten composition or document as distinguished from a printed copy 2 : writing as opposed to print

²sentinel *vt* **-neled** *or* **-nelled; -nel·ing** *or* **-nel·ling** (1593) 1 : to watch over as a sentinel 2 : to furnish with a sentinel 3 : to post as sentinel

sen·try \'sen-trē\ *n, pl* **sentries** [perh. fr. obs. *sentry* (sanctuary, watch-tower)] (1632) : GUARD, WATCH; *esp* : a soldier standing guard at a point of passage (as a gate)

sentry box *n* (ca. 1728) : a shelter for a sentry on his post

se·pal \'sēp-əl, 'sep-\ *n* [NL *sepalum*, fr. *sepa-* (fr. Gk *skepē* covering) + *-lum* (as in *petalum* petal); akin to Lith *kepurė* head covering] (1821) : one of the modified leaves comprising a calyx — see FLOWER illustration

se·pal·oid \-ə-ˌlȯid\ *adj* (1830) : resembling or functioning as a sepal

·sep·al·ous \'sep-ə-ləs\ *adj comb form* [*sepal*] : having (such or so many) sepals ⟨gamosepalous⟩

sep·a·ra·ble \'sep-(ə-)rə-bəl\ *adj* [ME, fr. L *separabilis*, fr. *separare*] (14c) 1 : capable of being separated or dissociated 2 *obs* : causing separation — **sep·a·ra·bil·i·ty** \ˌsep-(ə-)rə-ˈbil-ət-ē\ *n* — **sep·a·ra·ble·ness** *n* — **sep·a·ra·bly** \-blē\ *adv*

¹sep·a·rate \'sep-(ə-)ˌrāt\ *vb* **-rat·ed; -rat·ing** [ME *separaten*, fr. L *separatus*, pp. of *separare*, fr. *se-* apart + *parare* to prepare, procure — more at SECEDE, PARE] *vt* (15c) 1 **a** : to set or keep apart : DISCONNECT, SEVER **b** : to make a distinction between : DISCRIMINATE, DISTINGUISH ⟨~ religion from magic⟩ **c** : SORT ⟨~ mail⟩ **d** : to disperse in space or time : SCATTER ⟨widely *separated* homesteads⟩ 2 *archaic* : to set aside for a special purpose : CHOOSE, DEDICATE 3 : to part by a legal separation: **a** : to sever conjugal ties with **b** : to sever contractual relations with : DISCHARGE ⟨*separated* from the army⟩ 4 : to block off : SEGREGATE 5 **a** : to isolate from a mixture : EXTRACT ⟨~ cream from milk⟩ **b** : to divide into constituent parts ~ *vi* 1 : to become divided or detached 2 **a** : to sever an association : WITHDRAW **b** : to cease to live together as man and wife 3 : to go in different directions 4 : to become isolated from a mixture

syn SEPARATE, PART, DIVIDE, SEVER, SUNDER, DIVORCE mean to become or cause to become disunited or disjointed. SEPARATE may imply any of several causes such as dispersion, removal of one from others, or presence of an intervening thing; PART implies the separating of things or persons in close union or association; DIVIDE implies separating into pieces or sections by cutting or breaking; SEVER implies violence esp. in the removal of a part or member; SUNDER suggests violent rending or wrenching apart; DIVORCE implies separating two things that commonly interact and belong together.

²sep·a·rate \'sep-(ə-)rət\ *adj* (1600) 1 **a** *archaic* : SOLITARY, SECLUDED **b** : IMMATERIAL, DISEMBODIED **c** : set or kept apart : DETACHED 2 **a** : not shared with another : INDIVIDUAL ⟨~ rooms⟩ **b** *often cap* : estranged from a parent body ⟨~ churches⟩ 3 **a** : existing by itself : AUTONOMOUS **b** : dissimilar in nature or identity *syn* see DISTINCT — **sep·a·rate·ly** \-(ə-)rət-lē, 'sep-ərt-lē\ *adv* — **sep·a·rate·ness** \-(ə-)rət-nəs\ *n*

³sep·a·rate \'sep-(ə-)rət\ *n* (1892) 1 : OFFPRINT 2 : an article of dress designed to be worn interchangeably with others to form various costume combinations — usu. used in pl.

sep·a·ra·tion \ˌsep-ə-ˈrā-shən\ *n* (15c) 1 : the act or process of separating : the state of being separated 2 **a** : a point, line, or means of division **b** : an intervening space : GAP 3 **a** : cessation of cohabitation between husband and wife by mutual agreement or judicial decree **b** : termination of a contractual relationship (as employment or military service)

sep·a·ra·tion·ist \-sh(ə-)nəst\ *n* (1882) : SEPARATIST

sep·a·rat·ism \'sep-(ə-)rət-ˌiz-əm\ *n* (1628) : a belief in, movement for, or state of separation (as schism, secession, or segregation)

sep·a·rat·ist \'sep-(ə-)rət-əst, 'sep-ə-ˌrāt-\ *n* (1608) : one that favors separatism: as **a** *cap* : one of a group of 16th and 17th century English Protestants preferring to separate from rather than to reform the Church of England **b** : an advocate of independence or autonomy for a part of a political unit (as a nation) **c** : an advocate of racial or cultural separation — **separatist** *adj* — **sep·a·ra·tis·tic** \ˌsep-(ə-)rə-ˈtis-tik\ *adj*

sep·a·ra·tive \'sep-ə-ˌrāt-iv, 'sep-(ə-)rət-\ *adj* (1592) : tending toward, causing, or expressing separation

sep·a·ra·tor \'sep-(ə-),rāt-ər\ *n* (1607) : one that separates; *esp* : a device for separating liquids of different specific gravities (as cream from milk) or liquids from solids

Se·phar·di \sə-ˈfärd-ē\ *n, pl* **Se·phar·dim** \-ˈfärd-əm\ [LHeb *sēphāradhi*, fr. *sēphāradh* Spain, fr. Heb, region where Jews were once exiled (Obad 1: 20)] (1851) : a member of the occidental branch of European Jews settling in Spain and Portugal and later in Greece, the Levant, England, the Netherlands, and the Americas; *also* : one of their descendants — **Se·phar·dic** \-ˈfärd-ik\ *adj*

¹se·pia \'sē-pē-ə\ *n* [L, cuttlefish, ink, fr. Gk *sēpia*; akin to Gk *sēpein* to make putrid, *sapros* rotten] (1821) 1 **a** : the inky secretion of a cuttlefish **b** : a brown melanin-containing pigment from the ink of cuttlefishes 2 : a print or photograph of a brown color resembling sepia 3 : a brownish gray to dark olive brown

²sepia *adj* (1827) 1 : made of or done in sepia 2 : of the color sepia

se·pi·o·lite \'sē-pē-ə-ˌlīt\ *n* [G *sepiolith*, fr. Gk *sēpion* cuttlebone (fr. *sēpia*) + G *-lith* -lite] (1854) : MEERSCHAUM 1

se·poy \'sē-ˌpȯi\ *n* [Pg *sipai*, fr. Hindi *sipāhī*, fr. Per, cavalryman] (1717) : a native of India employed as a soldier by a European power

sep·pu·ku \se-ˈpü-(ˌ)kü, 'sep-ə-ˌkü\ *n* [Jp] (1904) : HARA-KIRI

sep·sis \'sep-səs\ *n, pl* **sep·ses** \'sep-ˌsēz\ [NL, fr. Gk *sēpsis* decay, fr. *sēpein* to make putrid] (1876) : a toxic condition resulting from the spread of bacteria or their products from a focus of infection; *esp* : SEPTICEMIA

sept \'sept\ *n* [prob. alter. of *sect*] (1517) : a branch of a family; *esp* : CLAN

sep·tal \'sep-tᵊl\ *adj* (1839) : of or relating to a septum

sep·tate \'sep-ˌtāt\ *adj* (1846) : divided by or having a septum

Sep·tem·ber \sep-ˈtem-bər, səp-\ *n* [ME *Septembre*, fr. OF & OE, both fr. L *September* (seventh month), fr. *septem* seven — more at SEVEN] (bef. 12c) : the 9th month of the Gregorian calendar

sep·te·nar·i·us \ˌsep-tə-ˈnar-ē-əs, -ˈner-\ *n, pl* **-nar·ii** \-ē-ˌī, -ē-ˌē\ [L, fr. *septenarius* of seven, fr. *septeni* seven each, fr. *septem* seven] (1819) : a verse consisting of seven feet esp. in Latin prosody

sep·ten·de·cil·lion \(ˌ)sep-ˌten-di-ˈsil-yən\ *n, often attrib* [L *septendecim* seventeen (fr. *septem* seven + *decem* ten) + E *-illion* (as in *million*) — more at TEN] (ca. 1938) — see NUMBER table

sep·ten·ni·al \sep-ˈten-ē-əl\ *adj* [LL *septennium* period of seven years, fr. L *septem* + *-ennium* (as in *biennium*)] (1640) 1 : occurring or being done every seven years 2 : consisting of or lasting for seven years — **sep·ten·ni·al·ly** \-ə-lē\ *adv*

sep·ten·tri·on \sep-ˈten-trē-ˌän, -trē-ən\ *n* [ME, fr. MF, fr. L *septentrio*, sing. of *septentriones* the seven stars of Ursa Major or Ursa Minor, lit., the seven plow oxen, fr. *septem* seven + *trio* plow ox] *obs* (14c) : the northern regions : NORTH

sep·ten·tri·o·nal \-trē-ən-ᵊl\ *adj* (14c) : NORTHERN

sep·tet \sep-ˈtet\ *n* [G *septett*, fr. L *septem*] (1837) 1 : a musical composition for seven instruments or voices 2 : a group or set of seven; *esp* : the performers of a septet

sep·tic \'sep-tik\ *adj* [L *septicus*, fr. Gk *sēptikos*, fr. *sēpein* to make putrid — more at SEPIA] (1605) 1 : PUTREFACTIVE 2 : relating to, involving, or characteristic of sepsis

sep·ti·ce·mia \ˌsep-tə-ˈsē-mē-ə\ *n* [NL, fr. L *septicus* + NL *-emia*] (1866) : invasion of the bloodstream by virulent microorganisms from a local seat of infection accompanied esp. by chills, fever, and prostration — called also *blood poisoning*; compare SEPSIS — **sep·ti·ce·mic** \-ˈsē-mik\ *adj*

sep·ti·ci·dal \ˌsep-tə-ˈsīd-ᵊl\ *adj* [NL *septum* + L *-cidere* to cut, fr. *caedere* — more at CONCISE] (1819) : dehiscent longitudinally at or along a septum ⟨a ~ fruit⟩

septic sore throat *n* (1924) : an inflammatory sore throat caused by hemolytic streptococci and marked by fever, prostration, and toxemia

septic tank *n* (ca. 1902) : a tank in which the solid matter of continuously flowing sewage is disintegrated by bacteria

sep·tif·ra·gal \sep-ˈtif-ri-gəl\ *adj* [NL *septum* + L *frangere* to break — more at BREAK] (1819) : dehiscing by breaking away from the dissepiments ⟨a ~ pod⟩

sep·til·lion \sep-ˈtil-yən\ *n, often attrib* [F, fr. L *septem* + F *-illion* (as in *million*) — more at SEVEN] (1690) — see NUMBER table

sep·tu·a·ge·nar·i·an \(ˌ)sep-ˌt(y)ü-ə-jə-ˈner-ē-ən, ˌsep-tə-ˌwaj-ə-\ *n* [LL *septuagenarius* seventy years old, fr. L, of or containing seventy, fr. *septuageni* seventy each, fr. *septuaginta*] (1805) : a person whose age is in the seventies — **septuagenarian** *adj*

Sep·tu·a·ge·si·ma \ˌsep-tə-wə-ˈjes-ə-mə, -ˈjā-zə-\ *n* [ME, fr. LL, fr. L, fem. of *septuagesimus* seventieth, fr. *septuaginta* seventy; fr. its being approximately seventy days before Easter] (14c) : the third Sunday before Lent

Sep·tu·a·gint \sep-ˈt(y)ü-ə-jənt, 'sep-tə-wə-ˌjint\ *n* [LL *Septuaginta*, fr. L, seventy, irreg. fr. *septem* seven + *-ginta* (akin to L *viginti* twenty); fr. the approximate number of its translators — more at SEVEN, VIGESIMAL] (1633) : a pre-Christian Greek version of the Jewish Scriptures redacted by Jewish scholars and adopted by Greek-speaking Christians — **Sep·tu·a·gin·tal** \(ˌ)sep-ˌt(y)ü-ə-ˈjint-ᵊl, ˌsep-tə-wə-\ *adj*

sep·tum \'sep-təm\ *n, pl* **sep·ta** \-tə\ [NL, fr. L *saeptum* enclosure, fence, wall, fr. *saepire* to fence in, fr. *saepes* fence, hedge; akin to Gk *haimasia* stone wall] (1726) : a dividing wall or membrane esp. between bodily spaces or masses of soft tissue — compare DISSEPIMENT

¹sep·ul·cher *or* **sep·ul·chre** \'sep-əl-kər\ *n* [ME *sepulcre*, fr. OF, fr. L *sepulcrum*, *sepulchrum*, fr. *sepelire* to bury; akin to Gk *hepein* to care for, Skt *sapati* he serves] (13c) 1 : a place of burial : TOMB 2 : a receptacle for religious relics esp. in an altar

²sepulcher *or* **sepulchre** *vt* **-chered** *or* **-chred; -chering** *or* **-chring** \-k(ə-)riŋ\ (1591) 1 *archaic* : to place in or as if in a sepulcher : BURY 2 *archaic* : to serve as a sepulcher for

se·pul·chral \sə-ˈpəl-krəl *also* -ˈpül-\ *adj* (1615) 1 : MORTUARY 2 : suited to or suggestive of a sepulcher : FUNEREAL — **se·pul·chral·ly** \-krə-lē\ *adv*

sep·ul·ture \'sep-əl-ˌchu(ə)r\ *n* [ME, fr. OF, fr. L *sepultura*, fr. *sepultus*, pp. of *sepelire*] (13c) 1 : BURIAL 2 : SEPULCHER

se·qua·cious \si-ˈkwā-shəs\ *adj* [L *sequac-*, *sequax* inclined to follow, fr. *sequi*] (1643) 1 *archaic* : SUBSERVIENT, TRACTABLE 2 : intellectually servile — **se·qua·cious·ly** *adv* — **se·quac·i·ty** \-ˈkwas-ət-ē\ *n*

se·quel \'sē-kwəl *also* -ˌkwel\ *n* [ME, fr. MF *sequelle*, fr. L *sequela*, fr. *sequi* to follow — more at SUE] (15c) 1 : CONSEQUENCE, RESULT 2 **a** : subsequent development **b** : the next installment (as of a speech or story); *esp* : a literary or cinematic work continuing the course of a story begun in a preceding one

se·quela \si-ˈkwel-ə\ *n, pl* **se·quel·ae** \-ˈkwel-(ˌ)ē\ [NL, fr. L, sequel] (1793) 1 : an aftereffect of disease or injury 2 : a secondary result

¹se·quence \'sē-kwən(t)s, -ˌkwen(t)s\ *n* [ME, fr. ML *sequentia*, fr. LL, sequel, lit., act of following, fr. L *sequent-*, *sequens*, prp. of *sequi*] (14c) 1 : a hymn in irregular meter between the gradual and Gospel in masses for special occasions (as Easter) 2 : a continuous or connected series: as **a** : an extended series of poems united by a single theme (a sonnet ~) **b** : three or more playing cards usu. of the same suit in consecutive order of rank **c** : a succession of repetitions of a melodic phrase or harmonic pattern each in a new position **d** : a set of elements ordered as are the natural numbers **e** (1) : a succession of related shots or scenes developing a single subject or phase of a film story (2) : EPISODE 3 **a** : order of succession **b** : an arrangement of the tenses of successive verbs in a sentence designed to express a coherent relationship esp. between main and subordinate parts 4 **a** : CONSEQUENCE, RESULT **b** : a subsequent development 5 : continuity of progression

²sequence *vt* **se·quenced; se·quenc·ing** (1941) 1 : to arrange in a sequence 2 : to determine the sequence of chemical constituents (as amino-acid residues) in ⟨*sequenced* biological macromolecules⟩

se·quenc·er \'sē-kwən-sər, -ˌkwen(t)-sər\ *n* (1949) : any of various devices for arranging (as informational items or the events in the launching of a rocket) into or separating (as amino acids from protein) in a sequence

se-quen-cy \'sē-kwən-sē\ *n* [LL *sequentia*] (1818) : SEQUENCE 3a, 5
se-quent \'sē-kwənt\ *adj* [L *sequent-, sequens*, prp.] (1601) 1 : CONSECUTIVE, SUCCEEDING 2 : CONSEQUENT, RESULTANT
se-quen-tial \si-'kwen-chəl\ *adj* (1854) 1 : of, relating to, or arranged in a sequence : SERIAL ⟨~ file systems⟩ 2 : following in sequence 3 : relating to or based on a method of testing a statistical hypothesis that involves examination of a sequence of samples for each of which the decision is made to accept or reject the hypothesis or to continue sampling — **se-quen-tial-ly** \-'kwench-(ə-)lē\ *adv*
¹se-ques-ter \si-'kwes-tər\ *vt* **-tered; -ter-ing** \-t(ə-)riŋ\ [ME *sequestren*, fr. MF *sequestrer*, fr. LL *sequestrare* to surrender for safekeeping, set apart, fr. L *sequester* agent, depositary, bailee; akin to L *sequi* to follow] (14c) 1 a : to set apart : SEGREGATE b : SECLUDE, WITHDRAW 2 a : to seize esp. by a writ of sequestration b : to place (property) in custody esp. in sequestration 3 : to hold (as a metallic ion) in solution usu. by inclusion in an appropriate coordination complex
²sequester *n, obs* (1604) : SEPARATION, ISOLATION
se-ques-trate \'sēk-wəs-,trāt, 'sek-; si-'kwes-\ *vt* **-trat-ed; -trat-ing** [LL *sequestratus*, pp. of *sequestrare*] (1513) : SEQUESTER
se-ques-tra-tion \,sēk-wəs-'trā-shən, ,sek-; (,)sē-,kwes-\ *n* (15c) 1 : the act of sequestering : the state of being sequestered 2 a : a legal writ authorizing a sheriff or commissioner to take into custody the property of a defendant who is in contempt until he complies with the orders of a court b : a deposit whereby a neutral depositary agrees to hold property in litigation and to restore it to the party to whom it is adjudged to belong
se-ques-trum \si-'kwes-trəm\ *n, pl* **-trums** *also* **-tra** \-trə\ [NL, fr. L, legal sequestration; akin to L *sequester* bailee] (1831) : a fragment of dead bone detached from adjoining sound bone
se-quin \'sē-kwən\ *n* [F, fr. It *zecchino*, fr. *zecca* mint, fr. Ar *sikkah* die, coin] (1582) 1 : an old gold coin of Italy and Turkey 2 : a small plate of shining metal or plastic used for ornamentation esp. on clothing
se-quined *or* **se-quinned** \-kwənd\ *adj* (1582) : ornamented with or as if with sequins
se-qui-tur \'sek-wət-ər, -wə-,tú(ə)r\ *n* [L, it follows, 3d pers. sing. pres. indic. of *sequi* to follow — more at SUE] (1836) : the conclusion of an inference : CONSEQUENCE
se-quoia \si-'kwȯi-(y)ə\ *n* [NL, genus name, fr. *Sequoya* (George Guess)] (ca. 1866) : either of two huge coniferous California trees of the pine family that reach a height of over 300 feet: a : BIG TREE b : REDWOOD 3a
sera *pl of* SERUM
se-rac \sə-'rak, sā-\ *n* [F *sérac*, lit., a kind of white cheese, fr. ML *seracium* whey, fr. L *serum* whey — more at SERUM] (1860) : a pinnacle, sharp ridge, or block of ice among the crevasses of a glacier
se-ra-glio \sə-'ral-(,)yō, -'räl-\ *n, pl* **-glios** [It *serraglio* enclosure, seraglio, partly fr. ML *serraculum* enclosure, bar of a door, bolt, fr. LL *serare* to bolt; partly fr. Turk *saray* palace — more at SEAR] (1588) 1 : HAREM 1a 2 : a palace of a sultan
se-rai \sə-'rī\ *n* [Turk & Per; Turk *saray* mansion, palace, fr. Per *sarāī* mansion, inn] (1609) 1 : CARAVANSARY 2 : SERAGLIO 2
ser-al \'sir-əl\ *adj* (1858) : of, relating to, or constituting an ecological sere
se-ra-pe \sə-'räp-ē, -'rap-\ *n* [MexSp *sarape*] (1834) : a colorful woolen shawl worn over the shoulders esp. by Mexican men
ser-aph \'ser-əf\ *n, pl* **ser-a-phim** \-ə-,fim\ *or* **seraphs** [back-formation fr. *seraphim*] (1667) : SERAPHIM 2
ser-a-phim \'ser-ə-,fim\ *n pl* [LL *seraphim*, pl., seraphs, fr. Heb *sērāphīm*] (bef. 12c) 1 : an order of angels — see CELESTIAL HIERARCHY 2 *sing, pl* **seraphim** : one of the 6-winged angels standing in the presence of God — **se-raph-ic** \sə-'raf-ik\ *adj* — **se-raph-i-cal-ly** \-i-k(ə-)lē\ *adv*
Se-ra-pis \sə-'rā-pəs\ *n* [L, fr. Gk *Sarapis*] : an Egyptian god combining attributes of Osiris and Apis and having a widespread cult throughout Greece and Rome
Serb \'sərb\ *n* [Serb *Srb*] (1860) 1 : a native or inhabitant of Serbia 2 : SERBIAN 2 — Serb *adj*
Ser-bi-an \'sər-bē-ən\ *n* (1862) 1 : SERB 1 2 a : the Serbo-Croatian language as spoken in Serbia b : a literary form of Serbo-Croatian using the Cyrillic alphabet — **Serbian** *adj*
Ser-bo–Cro-atian \,sər-(,)bō-krō-'ā-shən\ *n* (1883) 1 : the Slavic language of the Serbs and Croats consisting of Serbian written in the Cyrillic alphabet and Croatian written in the Roman alphabet 2 : one whose native language is Serbo-Croatian — **Serbo–Croatian** *adj*
¹sere \'si(ə)r\ *adj* [ME, fr. OE *sēar* dry; akin to OHG *sōrēn* to wither, Gk *hauos* dry] (bef. 12c) 1 : being dried and withered 2 *archaic* : THREADBARE
²sere *n* [L *series* series] (1939) : a series of ecological communities formed in ecological succession
¹ser-e-nade \,ser-ə-'nād\ *n* [F *sérénade*, fr. It *serenata*, fr. *sereno* clear, calm (of weather), fr. L *serenus* serene] (1649) 1 a : a complimentary vocal or instrumental performance; *esp* : one given outdoors at night for a woman b : a work so performed 2 : an instrumental composition in several movements, written for a small ensemble, and midway between the suite and the symphony in style
²serenade *vb* **-nad-ed; -nad-ing** *vt* (1672) : to perform a serenade in honor of ~ *vi* : to play a serenade — **ser-e-nad-er** *n*
ser-e-na-ta \,ser-ə-'nät-ə\ *n* [It, serenade] (ca. 1724) : an 18th century secular cantata of a dramatic character usu. composed in honor of an individual or event
ser-en-dip-i-tous \,ser-ən-'dip-ət-əs\ *adj* (1943) : obtained or characterized by serendipity ⟨~ discoveries⟩ — **ser-en-dip-i-tous-ly** *adv*
ser-en-dip-i-ty \-'dip-ət-ē\ *n* [fr. its possession by the heroes of the Persian fairy tale *The Three Princes of Serendip*] (1754) : the faculty of finding valuable or agreeable things not sought for
¹se-rene \sə-'rēn\ *adj* [L *serenus*; akin to OHG *serawēn* to become dry, Gk *xēros* dry] (1503) 1 : AUGUST — used as part of a title ⟨His *Serene* Highness⟩ 2 : marked by or suggestive of utter calm and unruffled repose or quietude ⟨a ~ smile⟩ 3 a : clear and free of storms or unpleasant change ⟨~ skies⟩ b : shining bright and steady ⟨the moon, ~ in glory —Alexander Pope⟩ *syn* see CALM — **se-rene-ly** *adv* — **se-rene-ness** \-'rēn-nəs\ *n*
²serene *n* (1644) 1 : a serene condition or expanse (as of sky, sea, c light) 2 : SERENITY, TRANQUILLITY
se-ren-i-ty \sə-'ren-ət-ē\ *n* [ME, fr. MF *serenité*, fr. L *serenitat-, serenita* fr. *serenus* serene] (15c) : the quality or state of being serene
serf \'sərf\ *n* [F, fr. L *servus* slave, servant, serf — more at SERVE] (161 : a member of a servile feudal class bound to the soil and subject to th will of his lord — **serf-age** \'sər-fij\ *n* — **serf-dom** \'sərf-dəm, -təm\ *n*
serge \'sərj\ *n* [ME *sarge*, fr. MF, fr. (assumed) VL *sarica*, fr. L *seric* fem. of *sericus* silken — more at SERICEOUS] (14c) : a durable twille fabric having a smooth clear face and a pronounced diagonal rib on th front and the back
ser-gean-cy \'sär-jən-sē\ *n* (1670) : the function, office, or rank of sergeant
ser-geant \'sär-jənt\ *n* [ME, servant, attendant, sergeant, fr. MF *serger serjant*, fr. L *servient-, serviens*, prp. of *servire* to serve] (14c) 1 : SE GEANT AT ARMS 2 *obs* : an officer who enforces the judgments of court or the commands of one in authority 3 : a noncommissione officer ranking in the army and marine corps above a corporal ar below a staff sergeant and in the air force above an airman first class senior airman and below a staff sergeant; *broadly* : NONCOMMISSIONE OFFICER 4 : an officer in a police force ranking in the U.S. just belo captain or sometimes lieutenant and in England just below inspector
sergeant at arms (14c) : an officer of an organization (as a legislati body or court of law) who preserves order and executes commands
sergeant first class *n* (1948) : a noncommissioned officer in the arn ranking above a staff sergeant and below a master sergeant
sergeant fish *n* (ca. 1883) 1 : COBIA 2 : SNOOK 1
sergeant major *n, pl* **sergeants major** *or* **sergeant majors** (1802) 1 : noncommissioned officer in the army, air force, or marine corps servi as chief administrative assistant in a headquarters 2 : a noncommi sioned officer in the marine corps ranking above a first sergeant 3 : bluish green to yellow percoid fish (*Abudefduf saxatilis*) with bla vertical stripes on the sides that is widely distributed in the weste tropical Atlantic ocean
sergeant major of the army (1966) : the ranking noncommissione officer of the army serving as adviser to the chief of staff
sergeant major of the marine corps (ca. 1971) : the ranking noncommi sioned officer of the marine corps serving as adviser to the comma dant
ser-geanty \'sär-jənt-ē\ *n, pl* **-geant-ies** [ME *sergeantie*, fr. MF *sergent* fr. *sergent* sergeant] (15c) : any of numerous feudal services of a pe sonal nature by which an estate is held of the king or other lord distin from military tenure and from socage tenure
serg-ing \'sər-jiŋ\ *n* [*serge*] (ca. 1909) : the process of overcasting t raw edges of a piece of fabric (as a carpet) to prevent raveling
¹se-ri-al \'sir-ē-əl\ *adj* (1841) 1 : of, relating to, consisting of, or a ranged in a series, rank, or row ⟨~ order⟩ 2 : appearing in successi parts or numbers ⟨a ~ story⟩ 3 : belonging to a series maturing pe odically rather than on a single date ⟨~ bonds⟩ 4 : of, relating to, being music based on a series of tones in an arbitrary but fixed patte without regard for traditional tonality ⟨~ technique⟩ — **se-ri-al-** \-ə-lē\ *adv*
²serial *n* (1846) 1 a : a work appearing (as in a magazine or on tele sion) in parts at intervals b : one part of a serial work : INSTALLME 2 : a publication (as a newspaper or journal) issued as one of a conse utively numbered and indefinitely continued series
se-ri-al-ism \'sir-ē-ə-,liz-əm\ *n* (1963) : serial music; *also* : the theory practice of composing serial music
se-ri-al-ist \-ləst\ *n* (1846) 1 : a writer of serials 2 : a composer serial music
se-ri-al-ize \-,līz\ *vt* **-ized; -iz-ing** (1892) : to arrange or publish in seri form — **se-ri-al-iza-tion** \,sir-ē-ə-lə-'zā-shən\ *n*
serial number *n* (1896) : a number indicating place in a series and us as a means of identification
¹se-ri-ate \'sir-ē-,āt, -ē-ət\ *adj* [(assumed) NL *seriatus*, fr. L *series*] (184 : arranged in a series or succession — **se-ri-ate-ly** *adv*
²se-ri-ate \'sir-ē-,āt\ *vt* **-at-ed; -at-ing** (1872) : to arrange in a series
¹se-ri-a-tim \,sir-ē-'āt-əm, -'at-\ *adv* [ML, fr. L *series*] (1680) : in a serie
²seriatim *adj* (1871) : following seriatim
se-ri-ceous \sə-'rish-əs\ *adj* [LL *sericeus* silken, fr. L *sericum* silk gr ment, silk, fr. neut. of *sericus* silken, fr. Gk *sērikos*, fr. *Sēres*, an easte Asian people producing silk in ancient times] (ca. 1777) : finely pub cent ⟨~ leaf⟩
ser-i-cin \'ser-ə-sən\ *n* [ISV, fr. L *sericum* silk] (ca. 1868) : a gelatino protein that cements the two fibroin filaments in a silk fiber
seri-cul-ture \'ser-ə-,kəl-chər\ *n* [L *sericum* silk + E *culture*] (185 : the production of raw silk by raising silkworms — **seri-cul-tur-** \-'kəlch-(ə)-rəl\ *adj* — **seri-cul-tur-ist** \-rəst\ *n*
se-ries \'si(ə)r-(,)ēz\ *n, pl* **series** *often attrib* [L, fr. *serere* to join, li together; akin to L *sort-, sors* lot, Gk *eirein* to string together, *horm* chain, necklace] (1611) 1 a : a number of things or events of t same class coming one after another in spatial or temporal successi ⟨a concert ~⟩ ⟨the hall opened into a ~ of small rooms⟩ b : a set regularly presented television programs each of which is complete itself 2 : the indicated sum of a usu. infinite sequence of numbers a : the coins or currency of a particular country and period b : group of postage stamps in different denominations 4 : a successi of volumes or issues published with related subjects or authors, simil format and price, or continuous numbering 5 : a division of ro formations that is smaller than a system and comprises rocks deposit during an epoch 6 : a group of chemical compounds related in co position and structure 7 : an arrangement of the parts of or elemer in an electric circuit whereby the whole current passes through ea part or element without branching 8 : a set of vowels connected ablaut (as *i, a, u* in *ring, rang, rung*) 9 a : a number of games (as baseball) played usu. on consecutive days between two teams ⟨in to for a 3-game ~⟩ b : WORLD SERIES 10 : a group of successive coor nate sentence elements joined together ⟨an *a*, *b*, and *c* ~⟩ 11 : SC SERIES 12 : three consecutive games in bowling — **in series** : in a ser arrangement
series wound *n* (ca. 1909) : a winding in which the armature coil a the field-magnet coil are in series with the external circuit — **se-rie wound** \,sir-ēz-'waünd\ *adj*

eminence or preference; SPECIFIC implies a quality or character distinguishing a kind or a species; PARTICULAR stresses the distinctness of something as an individual; INDIVIDUAL implies unequivocal reference to one of a class or group.

²special *n* (ca.1909) 1 : something (as a television program) that is not part of a regular series 2 : one that is used for a special service or occasion ⟨caught the commuter ~ to work⟩

special assessment *n* (1875) : a specific tax levied on private property to meet the cost of public improvements that enhance the value of the property

special delivery *n* (1886) : expedited messenger delivery of mail matter for an extra fee

special district *n* (1950) : a political subdivision of a state established to provide a single public service (as water supply or sanitation) within a specific geographical area

special drawing rights *n* (1967) : a means of exchange used by governments to settle their international indebtedness

special effects *n pl* (1944) : visual or sound effects introduced into a motion picture or a taped television production during laboratory processing

Special Forces *n pl* (1962) : a branch of the army composed of men specially trained in guerrilla warfare

special handling *n* (1928) : the handling of parcel-post or fourth-class mail as first-class but not as special-delivery matter for an extra postal fee

special interest *n* (1910) : a person or group seeking to influence legislative or government policy to further often narrowly defined interests; *esp* : LOBBY

spe·cial·ism \'spesh-ə-ˌliz-əm\ *n* (1856) 1 : specialization in an occupation or branch of learning 2 : a field of specialization : SPECIALTY

spe·cial·ist \'spesh-(ə-)ləst\ *n* (1856) 1 : one who devotes himself to a special occupation or branch of learning 2 : any of four enlisted ranks in the army corresponding to the grades of corporal through sergeant first class — **specialist** *or* **spe·cial·is·tic** \ˌspesh-ə-'lis-tik\ *adj*

spe·ci·al·i·ty \ˌspesh-ē-'al-ət-ē\ *n, pl* **-ties** (15c) 1 : a special mark or quality 2 : a special object or class of objects 3 a : a special aptitude or skill b : a particular occupation or branch of learning

spe·cial·iza·tion \ˌspesh-(ə-)lə-'zā-shən\ *n* (1843) 1 : a making or becoming specialized 2 a : structural adaptation of a body part to a particular function or of an organism for life in a particular environment b : a body part or an organism adapted by specialization

spe·cial·ize \'spesh-ə-ˌliz\ *vb* **-ized; -iz·ing** *vt* (1613) 1 : to make particular mention of : PARTICULARIZE 2 : to apply or direct to a specific end or use ⟨*specialized* his study⟩ ~ *vi* 1 : to concentrate one's efforts in a special activity or field 2 : to undergo specialization; *esp* : to change adaptively ⟨the sloth became highly *specialized* in the course of evolution⟩

specialized *adj* (1853) 1 : designed or fitted for one particular purpose or occupation ⟨~ personnel⟩ 2 : characterized by or exhibiting biological specialization; *esp* : highly differentiated esp. in a particular direction or for a particular end

special jury *n* (1730) : a jury chosen by the court on request from a list of better educated or presumably more intelligent prospective jurors for a case involving complicated issues of fact or serious felonies — called also *blue-ribbon jury*

special pleading *n* (1684) 1 : the allegation of special or new matter to offset the effect of matter pleaded by the opposite side and admitted, as distinguished from a direct denial of the matter pleaded 2 : misleading argument that presents one point or phase as if it covered the entire question at issue

special theory of relativity (1924) : RELATIVITY 3a

spe·cial·ty \'spesh-əl-tē\ *n, pl* **-ties** [ME *specialte*, fr. MF *especialté*, fr. LL *specialitat-, specialitas*, fr. L *specialis* special] (14c) 1 : a distinctive mark or quality 2 a : a special object or class of objects: as (1) : a legal agreement embodied in a sealed instrument (2) : a product of a special kind or of special excellence ⟨fried chicken was father's ~⟩ b : the state of being special, distinctive, or peculiar 3 : something in which one specializes

spe·ci·a·tion \ˌspē-s(h)ē-'ā-shən\ *n* (ca. 1900) : the process of biological species formation — **spe·ci·ate** \'spē-s(h)ē-ˌāt\ *vi* — **spe·ci·a·tion·al** \ˌspē-s(h)ē-'ā-shnəl, -shən²l\ *adj*

¹spe·cie \'spē-shē, -sē\ *n* [fr. *in specie*, fr. L, in kind] (1617) : money in coin — **in specie** : in the same or like form or kind ⟨ready to return insult *in specie*⟩; *also* : in coin

²specie *n* [back-formation fr. *species* (taken as a pl.)] *substand* (1711) : SPECIES

¹spe·cies \'spē-(ˌ)shēz, -(ˌ)sēz\ *n, pl* **species** [L, appearance, kind, species — more at SPY] (1551) 1 a : a class of individuals having common attributes and designated by a common name; *specif* : a logical division of a genus or more comprehensive class b : KIND, SORT c : the human race : human beings — often used with *the* ⟨survival of the ~ in the nuclear age⟩ d (1) : a category of biological classification ranking immediately below the genus or subgenus, comprising related organisms or populations potentially capable of interbreeding, and being designated by a binomial that consists of the name of a genus followed by a Latin or latinized uncapitalized noun or adjective agreeing grammatically with the genus name (2) : an individual or kind belonging to a biological species e : a particular kind of atomic nucleus, atom, molecule, or ion 2 : the consecrated eucharistic elements of the Roman Catholic or Eastern Orthodox Eucharist 3 a : a mental image; *also* : a sensible object b : an object of thought correlative with a natural object

²species *adj* (1899) : belonging to a biological species as distinguished from a horticultural variety ⟨a ~ rose⟩

spe·cies·ism \'spē-shē-ˌziz-əm, -sē-\ *n* [*species* + *-ism* (as in *racism*)] (1973) : prejudice or discrimination based on species; *esp* : discrimination against animals

¹spe·cif·ic \spi-'sif-ik\ *adj* [LL *specificus*, fr. L *species*] (1631) 1 a : constituting or falling into a specifiable category b : sharing or being those properties of something that allow it to be referred to a particular category 2 a : restricted to a particular individual, situation, relation, or effect ⟨a disease ~ to horses⟩ b : exerting a distinctive influence (as on a body part or a disease) ⟨~ antibodies⟩ 3 : free from ambiguity : ACCURATE ⟨a ~ statement of faith⟩ 4 : of, relating to, or constituting a species and esp. a biologic species 5 a : being any of various arbitrary physical constants and esp. one relating a quantitative attribute to unit mass, volume, or area b : imposed at a fixed rate per unit (as of weight or count) ⟨~ import duties⟩ — compare AD VALOREM *syn* see SPECIAL, EXPLICIT — **spe·cif·i·cal·ly** \-i-k(ə-)lē\ *adv*

²specific *n* (1661) 1 a : something peculiarly adapted to a purpose or use b : a drug or remedy having a specific mitigating effect on a disease 2 a : a characteristic quality or trait b : DETAIL, PARTICULARS — usu. used in pl. ⟨haggling over the legal and financial ~s of independence —*Time*⟩ c *pl* : SPECIFICATION 2a

spec·i·fi·ca·tion \ˌspes-(ə-)fə-'kā-shən\ *n* (1615) 1 : the act or process of specifying 2 a : a detailed precise presentation of something or of a plan or proposal for something — usu. used in pl. b : a statement of legal particulars (as of charges or of contract terms); *also* : a single item of such statement c : a written description of an invention for which a patent is sought

specific epithet *n* (1947) : the Latin or latinized noun or adjective that follows the genus name in a taxonomic binomial

specific gravity *n* (1666) : the ratio of the density of a substance to the density of some substance (as pure water or hydrogen) taken as a standard when both densities are obtained by weighing in air

specific heat *n* (1832) 1 : the ratio of the quantity of heat required to raise the temperature of a body one degree to that required to raise the temperature of an equal mass of water one degree 2 : the heat in calories required to raise the temperature of one gram of a substance one degree centigrade

specific impulse *n* (1947) : the thrust produced per unit rate of consumption of the propellant that is usu. expressed in pounds of thrust per pound of propellant used per second and that is a measure of the efficiency of a rocket engine

spec·i·fic·i·ty \ˌspes-ə-'fis-ət-ē\ *n* (1876) : the quality or condition of being specific: as a : the condition of being peculiar to a particular individual or group of organisms ⟨host ~ of a parasite⟩ b : the condition of participating in or catalyzing only one or a few chemical reactions ⟨the ~ of an enzyme⟩

specific performance *n* (1873) 1 : the performance of a legal contract strictly or substantially according to its terms 2 : an equitable remedy enjoining specific performance

spec·i·fy \'spes-ə-ˌfī\ *vt* **-fied; -fy·ing** [ME *specifien*, fr. MF *specifier*, fr. LL *specificare*, fr. *specificus*] (14c) 1 : to name or state explicitly or in detail 2 : to include as an item in a specification — **spec·i·fi·able** \-ˌfī-ə-bəl\ *adj* — **spec·i·fi·er** \-ˌfī(-ə)r\ *n*

spec·i·men \'spes-(ə-)mən\ *n* [L, fr. *specere* to look at, look — more at SPY] (1610) 1 : an item or part typical of a group or whole 2 a : something that obviously belongs to a particular category but is noticed by reason of an individual distinguishing characteristic b : PERSON, INDIVIDUAL ⟨he's a tough ~⟩ *syn* see INSTANCE

spe·ci·os·i·ty \ˌspē-shē-'äs-ət-ē\ *n* (1608) : the quality or state of being specious : SPECIOUSNESS

spe·cious \'spē-shəs\ *adj* [ME, fr. L *speciosus* beautiful, plausible, fr. *species*] (15c) 1 *obs* : SHOWY 2 : having deceptive attraction or allure 3 : having a false look of truth or genuineness : SOPHISTIC — **spe·cious·ly** *adv* — **spe·cious·ness** *n*

¹speck \'spek\ *n* [ME *specke*, fr. OE *specca*] (bef. 12c) 1 : a small discoloration or spot esp. from stain or decay 2 : a very small amount : BIT 3 : something marked or marred with specks — **specked** \'spekt\ *adj*

²speck *vt* (1580) : to produce specks on or in

¹speck·le \'spek-əl\ *n* [ME; akin to OE *specca*] (15c) : a little speck (as of color)

²speckle *vt* **speck·led; speck·ling** \-(ə-)liŋ\ (ca. 1570) 1 : to mark with speckles 2 : to be distributed in or on like speckles

speckled perch *n* (1888) : BLACK CRAPPIE

speckled trout *n* (1805) 1 : BROOK TROUT 2 : SPOTTED SEA TROUT

¹specs \'speks\ *n pl* [contr. of *spectacles*] (1807) : EYEGLASSES

²specs *n pl* [by contr.] (1942) : SPECIFICATIONS

spec·ta·cle \'spek-ti-kəl *also* -ˌtik-əl\ *n* [ME, fr. MF, fr. L *spectaculum*, fr. *spectare* to watch, fr. *spectus*, pp. of *specere* to look, look at — more at SPY] (14c) 1 a : something exhibited to view as unusual, notable, or entertaining; *esp* : an eye-catching or dramatic public display b : an object of curiosity or contempt ⟨made a ~ of herself⟩ 2 *pl* : GLASSES 3 : something (as natural markings on an animal) suggesting a pair of glasses

spec·ta·cled \-ti-kəld, -ˌtik-əld\ *adj* (1607) 1 : having or wearing spectacles 2 : having markings suggesting a pair of spectacles ⟨a ~ alligator⟩

¹spec·tac·u·lar \spek-'tak-yə-lər, spək-\ *adj* [L *spectaculum*] (1682) : of, relating to, or constituting a spectacle : STRIKING, SENSATIONAL ⟨a ~ display of fireworks⟩ — **spec·tac·u·lar·ly** *adv*

²spectacular *n* (1890) : something that is spectacular

spec·tate \'spek-ˌtāt\ *vi* **spec·tat·ed; spec·tat·ing** [back-formation fr. *spectator*] (1709) : to be present as a spectator (as at a sports event)

spec·ta·tor \'spek-ˌtāt-ər, spek-'\ *n* [L, fr. *spectatus*, pp. of *spectare* to watch] (1586) : one who looks on or watches — **spectator** *adj*

spec·ter *or* **spec·tre** \'spek-tər\ *n* [F *spectre*, fr. L *spectrum* appearance, specter, fr. *specere* to look, look at — more at SPY] (1605) 1 : a visible disembodied spirit : GHOST 2 : something that haunts or perturbs the mind : PHANTASM ⟨the ~ of hunger⟩

spec·ti·no·my·cin \ˌspek-tə-nō-'mīs-ᵊn\ *n* [NL, fr. *spectabilis* + *-in* + *-o-* + *-mycin*] (ca. 1964) : a white crystalline broad-spectrum antibiotic $C_{14}H_{24}N_2O_7$ produced by a bacterium (*Streptomyces spectabilis*) that is used clinically esp. in the form of its hydrochloride to treat gonorrhea

spec·tral \'spek-trəl\ *adj* (1815) 1 : of, relating to, or suggesting a spectre : GHOSTLY 2 : of, relating to, or made by a spectrum — **spec·tral·ly** \'spek-trə-lē\ *adv*

spectral line *n* (1902) : one of a series of linear images of the narrow slit of a spectrograph or similar instrument corresponding to a component of the spectrum of the radiation emitted by a particular source

spectro- *comb form* [NL *spectrum*] : spectrum ⟨*spectro*scope⟩

spec·tro·flu·o·rom·e·ter \'spek-(ˌ)trō-ˌflü(-ə)r-'äm-ət-ər\ *also* **spec·tro·flu·o·rim·e·ter** \-'im-\ *n* [*spectr-* + *fluorometer*] (1962) : a device for measuring and recording fluorescence spectra — **spec·tro·flu·o·ro·met·ric**

CAD Tool Integration For ASIC Design: An End-Users Perspective

Pankaj Kukkal, Masato Imaizumi*,
and Hideaki Kobayashi
Electrical and Computer Engineering
University of South Carolina
Columbia, South Carolina

Abstract

The increasing complexity of VLSI design and the demand for quick-turnaround ASICs has forced the designer to choose the best CAD tools available from different vendors and integrate them into a customized and comprehensive CAD system. Vendors have developed comprehensive but open systems, called CAD frameworks, to ease the process of CAD tool integration.

This paper describes the ASIC development process and issues relating to CAD tool integration. In addition, state-of-the-art CAD frameworks and their impact on ASIC designers are described.

I. Introduction

Application specific integrated circuits (ASIC) technology has brought a great revolution in VLSI system design. Before the advent of ASICs, systems were designed using a set of standard IC chips, whereas whole systems can now be designed into a single chip. ASICs are usually low volume products. Shorter design cycles are required to make the product cost competitive. Short design cycles imply increased dependence on CAD tools. To meet the demand of quick-turnaround ASICs, CAD tools have been developed to automate almost all the phases of ASIC design.

However, with increasing design complexities and design data, the flow of the design data through the CAD tools at various phases of the design cycle has become a cumbersome task for the ASIC designer. To relieve the designer of this burden, integrated CAD systems were developed. Initially, these systems were only a collection of tightly integrated tools. Adding and deleting tools was difficult, and usually these CAD systems were incompatible with each other. Such a closed environment is slow to evolve and the designer is restricted to a limited set of tools. Thus the concept of open design environments called CAD frameworks evolved. This concept is receiving wide attention in the area of computer aided design. A group of tool users, vendors, and system integrators, the CAD framework initiative (CFI), has set out to establish guidelines for a standard CAD framework. Thus we can look forward to a framework that encompasses all areas related to VLSI design such as specification capture, logic design, verification and testing.

II. Application Specific ICs

Definition

ASICs are ICs designed for a specific application, in contrast to standard ICs that are used in different applications. ASICs are a class of ICs that are in between the domain of software programmable generic components, for example micro-processors, and hardware programmable components, for example PLDs [KEUT,'89]. They can also be called custom-built ICs.

Characteristics

State-of-the-art electronics products such as digital signal processors, voice synthesizers, automatic focusing systems in cameras, etc. use ASICs. For prototypes, the circuits are designed by using a set of standard IC products but later for commercial products they are redesigned as an ASIC chip. System development cost is reduced by incorporating different components of the system, for example system controllers, RAMs, ROMs, PLDs, etc., into an ASIC. This technology also reduces the size of printed circuit boards thus reducing production cost. However, once systems are integrated into a chip, correcting even a small error, requires re-designing and re-fabricating the entire chip. Also, reduced node accessibility creates testing problems. Testing currently occupies 30-50 percent of the production cost of ASIC chips [LEUN,'88].

The ASIC designer uses a lot of CAD tools to reduce ASIC design time but has to sacrifice flexibility with regards to minimizing area and maximizing speed. A survey of ASICs designed for use by AT&T summarizes some common characteristics of ASICs [KEUT'89].

- *Control dominated* - Design of the control circuitry in the ASIC takes up a majority of circuit area and design time.
- *Arithmetic Structures* - Few large arithmetic circuits are present in an ASIC.
- *Speed/area* - Even though high speed and component density are achievable, most ASICs use under 10,000 logic (non-memory) transistors and operate at no more than 10 MHz.
- *Regularity of structure* - ASIC designers use as many regular structures for eg. library cells, gate arrays, etc. as possible.
- *Analog interfaces* - Many asynchronous and analog interfaces are used.

III. ASIC Design Styles and Methodology

Design Styles

Designers can employ various design styles to design ASICs. Some of the design styles and trade-offs involved are described below.

Gate arrays - In this style gates are pre-arranged, with space for channel routing. These gates are then interconnected by customizing metal layers. All levels of masks, except the metal interconnections, are predefined so that the wafer can largely be pre-fabricated. This prefabrication of wafers, called masterslices, is the main reason for fast turn-around time of prototype gate arrays. Due to its limited design freedom, the chip size of the gate array is typically two or three times that of a handcrafted design [LEUN,'88].

Sea of gates - This style is similar to gate arrays but has no

* M. Imaizumi is a Visiting Researcher to the Department of Electrical and Computer Engineering on leave from LSI Project, NKK Corporation, 2-6-3 Hitotsubashi, Chiyodaku, Tokyo 101, Japan.

0094-2898/91/0000/0363$01.00 © 1991 IEEE

predefined routing channels. So, sea-of-gates is also called "channel less gate array" and is more effective in area reduction than gate arrays.

Standard cell - Standard cells, for example, basic cells like inverters, NAND gates, Flip-Flops, etc., are pre-designed and are a part of a cell library. Placement and routing can both be customized. Usually all cells are of fixed height and a channel router is used to accomplish the interconnections. Pre-fabricated wafers are not used in this style.

PLDs - Programmable logic devices include PLAs, which are generated by automatic PLA generators and are usually used to implement system controllers.

Custom cells - Custom cells are designed to implement special functions that are not available in the library. If these custom cells are designed according to standard cell height specifications, then they can be added to the cell library and used with standard cells.

Silicon compilers - Once silicon compilers come of age the designer need only provide behavioral descriptions of the circuit and the corresponding hardware will be synthesized on silicon automatically. Presently, silicon compilers are used for generating megacells [LEUN,'88].

Usually, an ASIC is a mixture of modules designed using different design styles. For example, IBM has developed a design system that allows complete mixing of standard cell and gate array functions [LEUN,'88]. Thus we are moving towards an era where changing from one design style to another will be easy. This implies that designers will no longer be required to make tradeoffs between the various design styles. An integrated design environment will provide the designer with a complete solution to intermixing various styles.

Design methodology

Various aspects of VLSI design can be partitioned hierarchically into levels of abstraction. Design tools are used to implement the design at these levels. *Table 1* describes an example of hierarchical levels, levels of abstractions, and CAD tools required to implement the design at these levels. A methodology is a sequence of design steps that links the design process from specification capture to mask layout. There are basically two types of hierarchical design methodologies: top-down (see *Figure 1*) and bottom-up. In top-down design methodology, a design is first described in general terms at some high level of design abstraction. Designers then recursively decompose and elaborate the design. In the bottom-up method the most detailed parts are designed first and then global layout is determined by combining these parts. Today, most designers employ the top-down design approach.

IV. ASIC Development Process

The ASIC development process is divided into four major areas. The design process being an important part of the overall ASIC development process, is presented here in greater detail.

Specification Capture - During specification capture, the objective or goal of designing the chip is decided. The performance characteristics and functionality of the chip are set forth. All constraints regarding the environment (i.e. all external factors affecting the chip in any possible way) are precisely defined [CAVI,'90].

Design Synthesis and Verification - This stage of ASIC development encompasses all the design phases from system design to the final mask layout. It can be viewed as a process of successive transformations from one hierarchical level to another.

Different phases of design synthesis and verification for a top-down design approach are listed as follows:

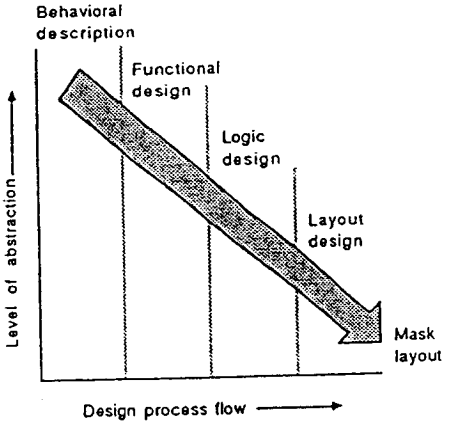


Figure 1. The top-down design process.

a. System is designed according to the specifications.
b. Decisions on fabrication process, for example the technology to be used, are made.
c. Functional partitioning of the system into modules is done.
d. Specifications for each module are set at some level of abstraction as shown in *Table1*.
e. Through a series of mapping and translation steps the design is taken down through the hierarchical levels, as shown in *Table 1*, to the logic design level.
f. Logic is verified.
g. The design is mapped down to the layout design level.
h. Inverse mapping (mapping from a lower level to a higher level) is done and the layout is verified for consistency with the logic design.
i. Logic and timing verification is performed at the layout level.
j. Mask layout for the chip is obtained in a standard format, for example CIF, and sent to the fabrication house.

Table 1. Hierarchical and abstraction levels.

Hierarchy Levels	Abstraction Levels	CAD Tools
System	Timing behavior, pin assignments	Flowcharts, Block-diagrams, High level languages
Architecture	Organization of functional blocks	HDL's, Floorplanning, Block diagrams
Register transfer	Developing specifications for functional modules	Synthesis, Simulation Verification, Test analysis
Logic	Boolean functions, Gate level circuits	Schematic entry, Simulation, Verification,
Transistor	Electrical properties of transistor circuits	SPICE, Timing verification
Layout	Geometric constraints	Layout editor, DRC and ERC programs, Netlist extractor, Placement and Routing tools

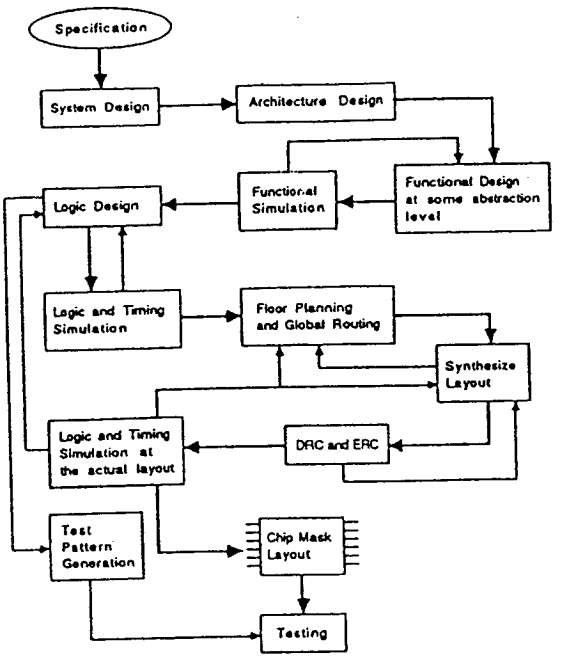


Figure 2. ASIC design flow.

Figure 2 depicts a typical ASIC design process. It can be seen from the figure that backtracking and iteration are required throughout the design process.

Fabrication - VLSI chips are fabricated by a complex series of about 100 or more steps. These steps create transistor parts, circuit elements, insulating layers, and metallized paths on silicon. Some major steps are thermal oxidation, lithography, etching, ion implantation, thermal redistribution, insulation, and metallization [STRO,'88].

Testing and Verification - Design verification techniques are used to qualify the ASIC to be marketed/supplied. At this stage the fabricated chip is tested for the specified external environment under accelerated stress conditions. Other tests to detect manufacturing defects are also carried out. Undetected errors in an ASIC can be very costly due to additional manufacturing costs and delay to market.

V. CAD Tools for ASIC Design

It is apparent from *Table 1* and *Figure 2* that a number of CAD tools are required to accomplish ASIC design tasks. These tools are classified as follows. (Examples for each class are taken from the UC Berkeley VLSI tool suite [SCOT,'85].)

Mapping Tools - These are used for schematic capture, interactive layout, graphical entry, text entry, placement and routing. For example, MAGIC layout editor is used for interactive layout, PEG is a text entry tool used for obtaining Boolean equations corresponding to a text that describes state diagrams.

Translation Tools - These are used for transforming files from one format to another to ensure compatibility between tools. For example, EXT2SIM is a tool for transforming the output of MAGIC layout editor to a format that is readable by simulator tools like ESIM and CRYSTAL.

Validation Tools - These tools check for violations of design rules. For example, the built-in Design Rule Checker (DRC) in the MAGIC layout editor concurrently checks for design rule (design rules depend on technology and fabrication process) violations while the designer is laying out the design. There are CAD tools that perform Electrical Rule Checking (ERC) and check for connectivity of all the elements of the design.

Verification Tools - Logic and timing simulation on designs are performed by these tools. For example, ESIM is used for logic verification, CRYSTAL and SPICE are used for Timing verification.

Optimization Tools - Optimization tools are used to optimize area and speed. Area can be optimized by either layout compaction or reduction in the logic required to implement the design. For example, PLEASURE is used for PLA folding, and ESPRESSO is used for minimizing Boolean equations. Speed optimization depends largely on the designers expertise.

Along with the CAD tools the ASIC designer requires cell libraries. Cell libraries are a collection of primitive components, which can be individual transistors, logic gates or entire subsystems. These libraries may be traditional standard cells, gate array building blocks, parameterized cells such as those synthesized by silicon compilers, or sophisticated cells generated by module generators [NEWT,'86].

Invoking and Executing CAD tools

Some basic terms required for this discussion are defined [FIDU,'90] as:

- *Process* is a specific combination of tools and/or other processes that perform a design function.
- *Task* is an abstraction of a design function, e.g., simulation. Tasks are performed by invoking specific processes.

Whatever design methodology/style is employed, an ASIC designer needs to perform a sequence of tasks by using a set of CAD tools and cell libraries. A designer needs to invoke and execute each CAD tool required for ASIC design. This implies that he needs to select an appropriate tool for the given task and then execute that tool so that it accomplishes exactly what needs to be done. To select a tool, the designer needs to know the various available tools for the given task and then chose the best. This can only be done if the designer knows the exact functionality of all the CAD tools. The designer also needs information on the versions of CAD tools available. Thus many decisions are required even before invoking a tool. To execute the selected tool the designer needs to know the exact syntax for tool invocation, and semantics of the tool.

The designer also has to manage all design data files related to the tools. As most tools are incompatible with regards to input and/or output file formats the designer needs to translate these files [BUSH,'89]. For example, to run a simulation on a layout, geometrical information needs to be extracted from the layout and then converted to a file format that can be input to a simulator. The whole design, from behavioral description to mask layout, needs to be steered by the designer using the CAD tools. The designer can be relieved of these painstaking tasks if the following information about all the tools is embedded in the system:

- *Data requirements*; type, access modes, format, etc.
- *Argument definitions*; format, required/optional, purpose, etc.
- *Tool commands*; syntax, arguments, purpose, effect, etc.
- *Resource requirements*; regarding CPU time, memory, etc.
- *Description of tool functions*.

In [DANI,'89] an object-oriented approach to build models of CAD tools has been proposed. By binding a CAD tool to its representative model, they create a CAD tool knowledge object

(CTKO). The CTKO represents the abilities of the tool to the designer, manages the low level programming details associated with the original tool, and provides a control mechanism between the tool and the designer.

CAD tool integration

CAD tools in the past decade have automated almost all the phases of the design process. The need of this decade is to integrate CAD tools into a design automation system. ASIC designers require an environment for integrating heterogeneous CAD tools while providing an open and distributed control mechanism. The need for CAD tool integration can be discussed with respect to four sub-topics as follows:

Design data management - With increasing design complexity, managing the input and output design files from the CAD tools is becoming extremely difficult. This is true especially when the data generated by CAD tools is enormous.

Number of CAD tools - CAD tool vendors are continuously supplying the designer with better CAD tools. Since many tools are incompatible in many ways, it takes a lot of time and effort to integrate them into a comprehensive design environment. This process takes a lot of time and effort especially when the CAD tools have different user-interfaces and file formats.

Complexity of CAD tools - Learning new CAD tools is not a trivial task; it involves a lot of time and effort. And if the designer does not continuously update his information about new and better tools he is left with a small set of CAD tools. To remain competitive the designer has to learn, master, and integrate new CAD tools to the existing CAD environment.

Tool invocation and execution - An integrated CAD environment will allow the designer to automatically invoke and execute CAD tools to perform a given task.

Three vital aspects of tool integration are visual, data, and control [SUNT,'90].

Visual integration - Visual integration implies that all the tools integrated should have the same look and feel. The interface between the designer and the tools should be homogeneous. This user interface must isolate the designer from the specifics of tool invocation and execution. The user must be able to choose from a list (usually icons) of tools or the execution environment must be able to choose one automatically based on the functionality of the tool. The interface should track and display the state of any on-going task including status of all related processes.

Data integration - Data integration can be classified as data linkage, data interchange, and data sharing as shown in *Figure 3*.

- *Data linkage* means establishing semantic relationships between pieces of information maintained by different tools. For example, Sun Microsystem's NSE Link Service 1.0 provides limited capability of this type [SUNT,'90].
- *Data interchange* means transferring information between tools in some mutually agreed upon representation. Examples of data interchange mechanism include window cut and paste, data import/export. Case Data Interchange Format (CDIF), is an example of a standard representation [SUNT,'90].
- *Data sharing* means that tools directly access data stored in a mutually accessible place. The Mentor Graphics' data repository approach is an example of data sharing that involves a common database schema and storage of data under a common data management system [WINK,'90].

Control integration - Control integration can be classified as interprocess control and meta control as shown in *Figure 4*.

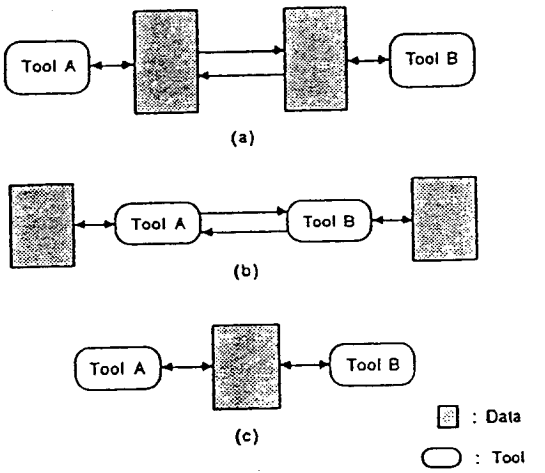


Figure 3. Data integration. (a) Data linkage. (b) Data interchange. (c) Data sharing.

- *Interprocess control* means that a tool can cause another tool to perform some action.
- *Meta control* means causing a sequence of tool invocations and requisite data inter-changes to accomplish some specific task automatically. Meta control can be implemented in terms of interprocess control, in which an 'agent' tool co-ordinates actions performed by other tools.

VI. The CAD Framework

We have already seen the need for CAD tool integration. This need has forced tool developers to create suites of tools that shield the designer from as much lower level detail as possible. Traditionally these suites were tightly integrated into a design environment [SIEW,'83] [BROW,'83]. CAD tools were bound into a monolithic entity. A problem with this approach was that tools were difficult to add or delete. This led to the creation of CAD Frameworks that are intended to be open architectures that support a large population of heterogeneous tools. CAD frameworks may be used to configure a set of VLSI tools and to develop appropriate interfaces to support schematic capture, simulation, timing verification, and test generation for ASIC design [HARR,'90]. It should act like a conduit between the

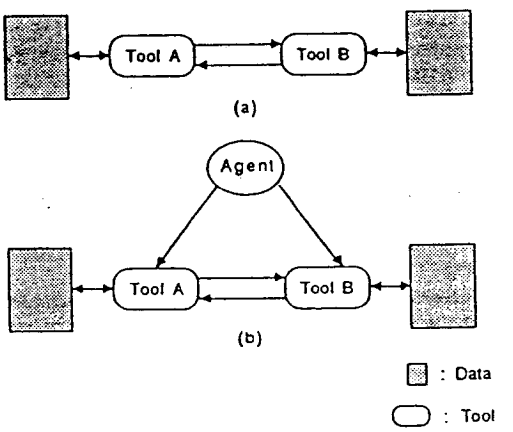


Figure 4. Control integration. (a) Interprocess control. (b) Meta control.

designer and the tools, matching the abilities of the tools to the needs of the designer. A framework should allow various tools to co-operate and work interactively with each other and with the designer. The designer need not learn all the subtle details of any tool; instead, the tools should present their general abilities to the designer as accurately and concisely as possible.

A generic CAD framework with its major components is shown in *Figure 5*. At a very high level the CAD framework can be viewed as having five major components described below [CAVI,'90]:

- A common user-interface which provides a consistent, graphical, and natural front end to the CAD tools.
- A design database containing design and library information which could be a centralized or distributed database.
- A design management database containing information on revisions, relationships, access authorizations, methodologies, and tools.
- A design data and process manager which utilizes the design management database to control design information and design process.
- CAD tools for all the phases of VLSI design.

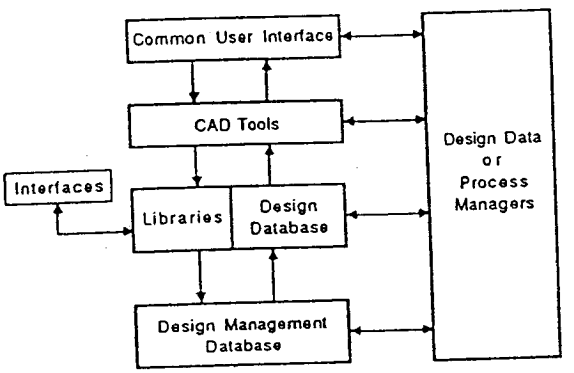


Figure 5. A generic CAD framework.

Characteristics

Some important characteristics of CAD frameworks that directly effect the designer are summarized in this section.

A framework supports multiple users, thus the design can be divided among various designers. A distributed heterogeneous computing environment can be created thus allowing the designers to use tools that work under different computing environments.

The designer does not have to worry about the syntax of tool invocations and the semantics of tool commands since CAD frameworks will allow for automatic execution of tools. The designer is relieved of the burden of making available the proper data at the proper time to all the tools required in the design process. Once fully integrated into a framework all the tools will have the same look and feel making it easier for the designer to get accustomed to many different tools.

The designer can intermix design entry levels, for example hardware description languages (HDLs) and schematics, for specifying a design. CAD frameworks will allow for simulations of such intermixed design entries.

A framework supports different design styles and methodologies. It also allows an expert designer to record and save successful design methodologies, allowing less experienced designers to create good designs using the knowledge of an expert designer. CAD frameworks combine top-down design with bottom-up schematic entry oriented design approach; this lets the designer tackle extremely complex IC designs. The designer can make arbitrary changes to the design at any stage and the framework will take care of design consistency by automatically propagating the effects of the changes to all related files.

Design complexity makes it impossible for the designer to foresee all situations where incompatibilities occur between the physical structure and the functional specifications. CAD frameworks will keep track of various versions of complex designs through all the phases of the development cycle while ensuring data integrity. Therefore, the designer can easily backtrack through all his previous design steps. He can also track the evolution of the design.

The framework eliminates long error-prone translations of data files between CAD tools since all the databases of the individual tools are either linked together by a common database or there is just a single database for all the tools.

CAD frameworks can be used for concurrent hardware and software design. Thus it will be possible to test software code on simulated hardware, permitting changes in either or both, to better meet the needs of the system [ADAM,'90]. It decreases the total turn-around time of the design process.

Until now, most of the decisions at every phase of the design were a matter of experience. With increasing complexities of the design process, just one wrong decision can prove fatal. The designer needs some decision support at the earlier stages of the design process. For example, if the designer decides on a particular floorplan he can get information about the affect of that floorplan on the final layout area of chip. Such decision support is provided by the framework.

CAD frameworks will allow the designers to work at a higher level of abstraction by allowing for simulations at the behavioral level. Designers need not become expert users of a baffling array of complex CAD tools, but instead need only be concerned with higher level design tasks. Designers can rectify potential problems before they happen at lower levels of design.

Knowledge required for good designs, such as the fundamentals of material properties, fabrication processes, device characteristics, microelectronics and circuit techniques, can also be built into the frameworks.

CAD Framework Initiative

Some vendors have come up with commercially available frameworks, for example Mentor Graphics' Falcon Framework, DEC's PowerFrame and Cadence's Design Framework II. Though there is general agreement over the basic definition of a framework, there are many different ways to implement the idea.

The CAD Framework Initiative (CFI), a body of CAD tool users, vendors, and system integrators was formed to supervise these ongoing framework efforts and lay down industry acceptable guidelines for design frameworks [CADF,'90]. CFI's purpose is to provide industry standard interfaces to CAD framework components, thus reducing the cost of combining multi-sourced CAD tools into an effective CAD system.

The CFI group defines a framework [CADF,'90] as:

"A software infra-structure that provides an environment where CAD tools are developed, integrated, and operated."

It further specifies that through a CAD framework, a user should be able to launch and manage tools; create, organize, and manage data; graphically view the entire design process; and perform design management tasks such as configuration management, version management, etc..

CFI and CAD tool integration

CFI has come up with a seven layer conceptual model [FRAM,'90] of a framework (*Figure 6*). Out of these seven layers two layers (Levels 4 and 5) refer specifically to CAD tools. Level 4 allows for tool management which means that the tools have to be managed within the context of the overall design process. It specifies that the frameworks should supply models and services that describe inputs, outputs, options, invocation sequences, control files, etc. to manage tools. Usually tools can be integrated into any CAD system but the integration software has to deal with data translation between the tools. Level 5 allows for easy integration by separating the data descriptions from the tool integration programs.

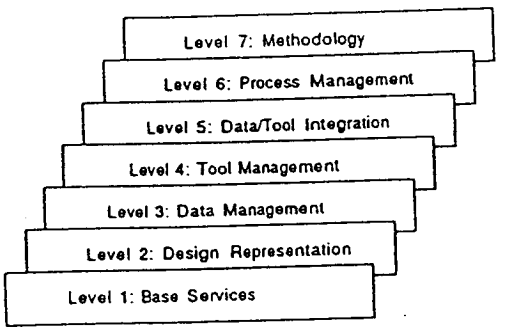


Figure 6. CFI's conceptual model of frameworks.

VII. Conclusion

In this paper we have identified the ASIC development process and CAD tools required for an ASIC design process. Then we have shown the evolution of computer aided VLSI design from the point where individual CAD tools were available to perform some of the design tasks, to a unified design environment called the CAD Framework. CAD frameworks not only integrate CAD tools into an open environment but also automate the design process. We have identified the impact of CAD frameworks on the designer and thus hope that this paper helps the ASIC designers to introduce the benefits of CAD frameworks into their existing design environment.

We look forward to CAD frameworks that will support a fully automated design process. The designer need only be involved in the process of specification capture and feeding behavioral/functional specifications to the CAD system. By the end of this decade we hope that there will be accepted standards for integrating not only CAD tools but tools that will encompass all other IC related areas such as manufacturing, control systems, process and device modelling, etc.

References

[ADAM,'90] Adams, Charlotte, "CAD Frameworks: Putting it all together", *Military and Aerospace Electronics*, pp. 33-38, July 1990.

[BROW,'83] Brown, H., et. al., "Palladio: An Exploratory Environment for Circuit Design", *IEEE Computer*, December 1983.

[BUSH,'89] Bushnell, M.L. and Director, S.W. , "Automated Design Tool Execution in the Ulysses Design Environment", *IEEE Transc. on Computer Aided Design*, vol. 8, no. 3, pp. 279-287, March 1989.

[CADF,'90] TCC-Approved Draft Proposal, "CAD Framework Users, Goals and Objectives", 1990.

[CAVI,'90] Cavin, R.K. and Hilbert, J.L. "Design of Integrated Circuits: Directions and Challenges", *Proceedings of the IEEE*, vol. 78, no. 2, pp. 418-435, Feb.1990.

[DANI,'89] Daniell, J. and Director, S.W., "An Object Oriented approach to CAD Tool Control within a Design Framework", *Proceedings IEEE 26th. Design Automation Conference*, pp. 197- 202, 1989.

[FIDU,'90] Fiduk, K.W., et al., "Design Methodology Management -A CAD framework Initiative Perspective", *Proceedings IEEE 27th. Design Automation Conference*, pp. 278-283,1990.

[FRAM,'90] Cadence's catalogue, "Framework Technology for the 1990s", 1990.

[HARR,'90] Harrison, D.S., et al., "Electronic CAD Frameworks", *Proceedings of the IEEE*, vol. 78, no. 2, pp. 393-417, Feb. 1990.

[KEUT,'89] Keuterzer, Kurt, "Three Competing Design Methodologies for ASICs: Architectural Synthesis, Logic Synthesis and Module generation", *Proceedings IEEE 26th. Design Automation Conference*, pp. 308-313, 1989.

[LEUN,'88] Leung, S.S., Fisher, P.D., and Shanblatt, M.A., " A conceptual framework for ASIC design", *Proceedings of the IEEE*, vol. 76, no. 7, pp. 741-755, July 1988.

[NEWT,'86] Newton, A. R., and Sangiovanni-Vincentelli, A. L., "Computer-Aided Design for VLSI circuits", *IEEE Computer*, pp. 38-60, April 1986.

[SCOT,'85] Scott, W. S., et. al., "1986 VLSI Tools: Still More Works by the Original Artists", *Report No. UCB/CSD 86/272*, Univ. of California, Berkeley, December 1985.

[SIEW, '83] Siewiorek, D. P. , et. al., "DEMETER-A Design Methodology and Environment", *Tech. report CMUCAD-83-5*, Electrical and Comp. Eng. Dept., Carnegie Mellon University, 1983.

[STRO,'88] Strojwas, A. J., and Director, S. W.,"VLSI: linking design and manufacturing", *IEEE Spectrum*, pp. 24-28, Oct. 1988.

[SUNT,'90] "CASE Integration Frameworks", *SunTech Journal*, Nov. 1990.

[WINK,'90] Winkler, E., "EDA Firms Find Several Roads to Frameworks", *Electronic News*, June 25, 1990.

INTERNATIONAL JOURNAL OF COMPUTER AIDED VLSI DESIGN

VOLUME 1
NUMBER 4

SPECIAL ISSUE: PART 2
VLSI CAD IN JAPAN

EDITOR
GEORGE W. ZOBRIST

ASSOCIATE EDITORS
ED CERNY
HIDEAKI KOBAYASHI
PAOLO PRINETTO
HARRY TYRER

ABLEX PUBLISHING CORPORATION
NORWOOD, NEW JERSEY

International

Journal of Computer Aided VLSI Design

Volume 1, Number 4, 1989

SPECIAL ISSUE: Part II

VLSI CAD IN JAPAN

Hideaki Kobayashi, Guest Editor

CONTENTS

Regular Contribution

INTERNATIONAL JOURNAL OF COMPUTER AIDED VLSI DESIGN

International Journal of Computer Aided VLSI Design is published quarterly by Ablex Publishing Corporation, 355 Chestnut Street, Norwood, New Jersey 07648

Volume 1, 1989

Individual Subscription: $32.50

Institutional Subscription: $85.00

An additional charge of $12.00 will be made for overseas postage.

Printed in U.S.A. ISSN 1042-7988

International

Journal of Computer Aided VLSI Design

Volume 1, Number 4, 1989

SPECIAL ISSUE: Part II

VLSI CAD IN JAPAN

Hideaki Kobayashi, Guest Editor

CONTENTS

Regular Contribution

International Journal of Computer Aided VLSI Design 1, 351–355 (1989)

Part II
Guest Editorial

HIDEAKI KOBAYASHI

Application-specific integrated circuit (ASIC) users in Japan are mainly "set makers" (system designers) who are not experts in VLSI design. They use (TTL) transistor-transistor logic–based techniques to design logic circuits (or netlists). Test vectors often are not generated by ASIC users. Many circuits designed by ASIC users do not implement initial chip functions. This often results in problems associated with design responsibilities between ASIC users and semiconductor makers.

With advanced CAD tools and systems, the interface between ASIC users and semiconductor makers will be shifted to a higher functional level. ASIC users can enter desired chip functions using optimum input forms such as truth tables, state diagrams, flowcharts, and hardware description languages (HDLs). ASIC users can also ensure initial chip functions without generating test vectors at a structural (circuit) level. Chips can be developed by ASIC users who are not familiar with circuit design or semiconductor technology.

Automatic translation between behavioral or functional inputs to logic circuits is provided by logic synthesis systems. There are two different types of approaches to automatic logic synthesis: algorithmic and rule-based. IF–THEN-type rules rather than algorithmic programming languages are used in the latter approach to synthesize logic circuits. These rules are extracted from expert ASIC designers who have been designing chips for many years.

Many Japanese semiconductor makers have developed in-house software for logic synthesis. A variety of commercial software for logic synthesis is also available in the Japanese market. Table 1 shows an example list of commercial software for logic synthesis. Table 2 shows an example list of in-house software for logic synthesis. These tables are translated from the article entitled "Logic Synthesis Software for ASIC Makes Design at the Functional Level Possible" by U. Kojima, published in *Nikkei Electronics*, November 1988 issue. Target applications of ASICs designed by these logic synthesis systems include audio visual machines, communications large-scale integration (LSI), digital signal processors, general-purpose microprocessors, micro program control, office automation, and peripheral LSI.

A list of other logic synthesis systems is provided in Table 3. These systems perform synthesis by translating a functional (algorithmic) description into a structural (data path and controller) description. They are compared in the second article of this special issue.

Table 1. Commercial Logic Synthesis Software

Name:	ZEPHCAD
Vendor:	Fujitsu Ltd.
Input form:	State diagram, Boolean expression, truth table
Design style:	CMOS gate array, CMOS standard cell
Name:	PARTHENON
Vendor:	NTT Data Communications Systems Corporation
Input form:	HDL (SFL)
Design style:	CMOS gate array, CMOS standard cell
Name:	Logic Synthesizer
Vendor:	LSI Logic Corporation
Input form:	State diagram, Truth table, Boolean expression, parameter
Design style:	CMOS gate array, CMOS standard cell
Name:	Finesse
Vendor:	Seattle Silicon Corporation
Input form:	Truth table, Boolean expression, state diagram
Design style:	CMOS standard cell
Name:	Logic Compiler
Vendor:	Silicon Compiler Systems Corporation
Input form:	Boolean expression, truth table, functional diagram, state diagram, HDL
Design style:	CMOS standard cell
Name:	SilcSyn
Vendor:	Silc Technologies, Inc.
Input form:	HDL (DDL)
Design style:	CMOS gate array, CMOS standard cell
Name:	Logic Consultant
Vendor:	Trimeter Technologies Corporation
Input form:	Boolean expression, netlist
Design style:	CMOS gate array
Name:	Design Compiler
Vendor:	Synopsys, Inc.
Input form:	HDL (Verilog), Boolean expression, truth table, netlist
Design style:	CMOS gate array, CMOS standard cell
Name:	State Machine Compiler
Vendor:	VLSI Technology, Inc.
Input form:	Boolean expression, truth table, parameter, state diagram, netlist, functional diagram, HDL (VHDL)
Design style:	CMOS gate array, CMOS standard cell

Table 2. In-House Logic Synthesis Software

Name:	FLORA
Company:	Fujitsu, Ltd.
Input form:	Functional diagram
Results:	Over 100 chips
Name:	ProLogic
Company:	Hitachi, Ltd.
Input form:	Structural and behavioral description of datapath and control circuits
Results:	Several samples under evaluation
Name:	LODES
Company:	Matsushita Electric Industrial Co., Ltd.
Input form:	Functional diagram, Boolean expression, truth table, HDL (HSL-FX)
Results:	Two chips for practical use
Name:	TL/C
Company:	Matsushita Electric Industrial Co., Ltd.
Input form:	Circuit diagram, Boolean expression, truth table, state diagram
Results:	Evaluation completed
Name:	LORES/EX
Company:	Mitsubishi Electric Corp.
Input form:	Functional diagram
Results:	Under evaluation
Name:	Fusion
Company:	NEC Corp.
Input form:	HDL (FDL)
Results:	Two chips
Name:	EXLOG
Company:	NEC Corp.
Input form:	HDL (FDL)
Results:	One chip
Name:	ANGEL
Company:	NEC Corp.
Input form:	HDL (HSL-FX)
Results:	Two chips
Name:	CLS
Company:	Oki Electric Industry Co., Ltd.
Input form:	HDL (HSL-FX)
Applications:	Microprocessors, DSP, control LSI
Results:	Five chips
Name:	Logic Synthesis System
Company:	Sharp Corp.
Input form:	Boolean expression, truth table, functional diagram
Results:	Under evaluation
Name:	Logic Synthesis Systems
Company:	Toshiba Corp.
Input form:	HDL (HHDL)
Results:	Thirty chips

Table 3. Other Logic Synthesis Software

Name:	CMU-DA System
Company:	Carnegie-Mellon University
Input form:	IPS language
Results:	Not available
Name:	VLSI Design Automation Assistant
Company:	AT&T Bell Laboratories
Input form:	ISPS language
Results:	Several digital systems
Name:	ALERT
Company:	IBM Watson Research Center
Input form:	Iverson's APL
Results:	Not available
Name:	Flamel
Company:	Stanford University
Input form:	Pascal
Results:	Not available
Name:	KBSC
Company:	Ricoh–International Chip Corporation
Input form:	Flowchart
Results:	Ten chips

The first article is "Unified Hardware Description Language and Its Support Tools" from Fujitsu Laboratories Ltd. A new hardware description language called UHDL (Unified Hardware Description Language) is proposed to describe asynchronous behavior and to synthesize large circuits. A hierarchical structure can be described by UHDL from several viewpoints, such as behavior, interface, and data paths.

The second article is "KBSC: A Knowledge-Based Approach to Automatic Logic Synthesis" from International Chip Corporation and Ricoh Company. A knowledge-based silicon compiler (KBSC) provides users with a front-end graphic interface to automatic logic synthesis. Both data-path and control circuits are synthesized by using rules extracted from expert ASIC designers. KBSC's flowchart input form allows ASIC users to enter desired chip functions easily, and verify these functions to ensure functional correctness.

The third article is "Development and Evaluation of an Architecture Design System for Application-Specific Integrated Circuits" from Waseda University. In this article, automatic architecture synthesis for special-purpose integrated circuits is discussed. The authors suggest that their system can synthesize and modify an initial circuit from a given algorithm. Target architectures are small- to medium-scale integrated circuits and are used to implement algorithms for digital signal processing.

The precise analysis of submicron pattern imaging becomes more important as feature sizes decrease. The minimum feature size of a 16-Mbit dynamic random-access memory is approximately half-microns. The fourth article is "A Novel Fast Calculation Method for Partial Coherent Images with Optical Aberration and Its Application to Submicron Pattern Imaging" from

Hitachi Ltd. This method calculates images of isolated square and isolated elbow mask patterns with various optical aberrations. It also calculates image distortions with several types of aberrations simultaneously.

Hideaki Kobayashi Dr. Kobayashi received both M.S. and Ph.D. degrees in Electrical Engineering from Waseda University, Tokyo, Japan. He joined the Department of Electrical and Computer Engineering at the University of South Carolina in 1980, where he is currently an Associate Professor. He is also Director of the VLSI/AI Design Laboratory at the Center for Machine Intelligence located at the new Swearingen Engineering Center, which is one of the most advanced facilities of its kind in the world. His current research interests include VLSI architecture for AI applications and knowledge-based silicon compilation. In addition to teaching on campus, his VLSI courses also are taught throughout the United States via the National Technological University.

International Chip Corporation was established in 1985 in Columbia, South Carolina as culmination of over ten years of research by Dr. Kobayashi, one of the two principals in the firm who also serves as Chairman of the Board. Since 1986 the company has funded continuing research at the University of South Carolina in a private-public partnership with Carolina Research and Development Foundation. For-profit business applications and related product development are conducted at International Chip Corporation's corporate headquarters, within walking distance of the main campus of the University of South Carolina.

Dr. Kobayashi has published over 50 professional papers in leading United States, Japanese, and European journals. He serves as an Associate Editor of the International Journal of Computer Aided VLSI Design. He is frequently invited to speak at professional meetings and seminars relating to ASIC (application specific integrated circuit) design, VLSI CAD, knowledge-based systems, and other associated topics. He also has been the recipient of research grants from several sources, including the National Science Foundation and the Semiconductor Research Corporation. His career contributions have been recognized in several ways, including selection as one of the Outstanding Young Men of America for 1985. He is a member of IEEE and Eta Kappa Nu.

International Journal of Computer Aided VLSI Design 1, 377–390 (1989)

KBSC: A Knowledge-Based Approach to Automatic Logic Synthesis

HIDEAKI KOBAYASHI
International Chip Corporation

TERUMI SUEHIRO AND MASAHIRO SHINDO
Ricoh Company, Ltd.

A knowledge-based silicon compiler (KBSC) has been jointly developed by Ricoh Company and International Chip Corporation. KBSC provides designers with a front-end graphic interface to automatic logic synthesis. A rule-based expert system synthesizes both data-path and control circuits. KBSC automatically translates an algorithmic description in flowchart form into a logic circuit (netlist) that consists of previously registered cells. ASIC design by KBSC is compared with design by a senior engineer in terms of design time, chip performance, and gate count. An inference engine chip for real-time rule processing is used as a design example. KBSC is also compared with other logic synthesis systems.

ASIC automatic logic synthesis flowchart input form
rule based expert systems silicon compilation

1 INTRODUCTION

Cost per large-scale integrated (LSI) chip can be defined by development cost D, fabrication cost F, and total production volume N. Therefore, chip cost C can be estimated by $C = D/N + F(N)$, where F is the function of N. If LSI circuits are produced in large volume, the development cost per chip is negligible. LSI circuits of this type include memories and other standard products.

ASICs are produced in small volume to meet customers' needs. In the case of ASICs, however, it becomes important to reduce development cost or design time. The process of ASIC design is divided into functional, logic, and layout stages. To reduce time required for functional and logic design, it is important to

1. Provide an optimum input form to specify an initial chip function.
2. Verify an initial chip function to ensure functional correctness.
3. Automatically translate a verified chip function to a logic circuit or netlist.

Correspondence and requests for reprints should be sent to Hideaki Kobayashi, International Chip Corporation, AT & T Building, 1201 Main St., Suite 2000, Columbia, SC 29201.

■ Manuscript received November 15, 1988; Manuscript revised June 5, 1989.

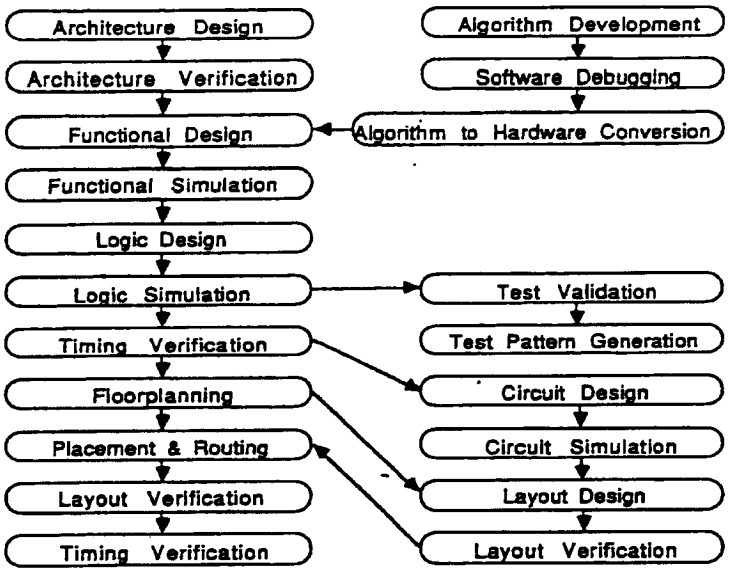


Figure 1. ASIC Development Flow.

Furthermore, to reduce time required for layout design, automatic translation of logic circuit (netlist) information to geometrical mask data is needed.

Many non-commercial logic synthesis systems [1–8] have been developed to translate automatically a behavioral or functional description into a logic circuit or chip layout. The concept of a knowledge-based silicon compiler (KBSC) was introduced in 1987 [9]. KBSC was developed jointly by Ricoh Company and International Chip Corporation (ICC). IF–THEN-type rules have been extracted from expert ASIC designers over several years at Ricoh and ICC, and stored in a knowledge base within KBSC. KBSC and other CAD tools are integrated into a comprehensive CAD system that provides designers with optimum input forms, such as flowcharts, state-transition equations, and functional module diagrams for cell-based design.

2 DESIGN METHODOLOGY

A typical ASIC development flow is shown in Figure 1. First, an entire system function is partitioned into a set of subfunctions (functional blocks). These functional blocks are then defined by various input forms, Figure 2.

1. Boolean expression or truth table form is for combinatorial circuit design.

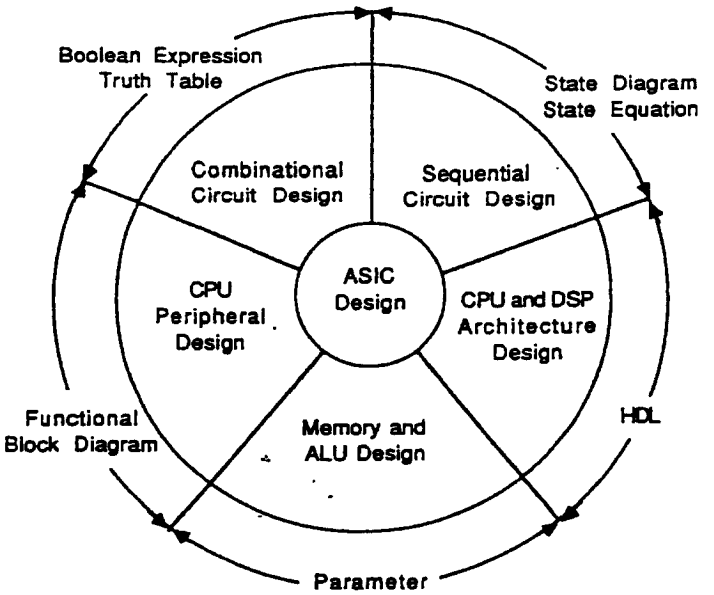


Figure 2. Input Forms for ASIC Design.

2. State-transition diagram or state-equation input form for sequential circuit design.
3. Parameter input form for memory and arithmetic and logic unit (ALU) design with variable address and data lengths.
4. Functional module input form using preregistered cells with bus compatibility for CPU peripheral LSI design.
5. Hardware description language (HDL) input form for CPU and digital signal processing (DSP) architecture design and performance evaluation.

Inputs 1 and 2 above require synthesis and optimization of logic. Logic synthesis and optimization are not required for input 3 since modules are generated by design rules and other constraints. Input 4 utilizes existing logic circuits corresponding to every module. Logic synthesis and optimization are not required since these circuits are already optimized. It is important to select an appropriate input form to design each functional block. The entire system design is completed by interconnecting these functional blocks.

KBSC's flowchart input form (Figure 4) is vital for "system" design that includes both data-path and control logic synthesis. KBSC flowcharts are useful for designers who are familiar with software programming as well as hardware system design. Flowchart-to-netlist conversion is achieved by an expert system where IF–THEN-type rules for logic synthesis are stored in a

knowledge base. These rules are extracted from expert ASIC designers at Ricoh and ICC. KBSC's cell-based design approach provides quick turn-around time and design flexibility. Logic circuits designed by KBSC contain previously registered cells, each with information on its own mask data. Therefore, only placement and routing of these cells need to be performed by layout tools.

Layout design is performed in a hierarchical manner. First, highest level cells (functional blocks) are placed on a chip. An optimum placement is achieved by our floor planner, based on information such as each block area (transistor count), *X/Y* dimension ratio, and a netlist between blocks. The floor planner computes relative *X/Y* coordinates, routing areas between blocks, and information about terminal locations in each block.

After initial placement (floor planning), automatic routing takes place. Our automatic place–route software can handle macrocells with four-sided terminals as well as cells with only upper and lower terminals. Savings of 20% or more of the chip area are achieved by this approach compared with our conventional place–route software for two-sided terminals. To achieve 100% routability, routing area is estimated by heuristics. Extra routing area is eliminated by compaction. Layout of lower-level blocks is performed in a similar manner.

A data base stores cells with circuit information and physical mask data. Each cell contains information on logic symbols for schematic capture, models for logic stimulation, terminal names and locations for automatic placement and routing, cell sizes, and physical mask data. These cells are used for automatic logic synthesis as well as schematic capture to design logic circuits.

3 SYSTEM CONFIGURATION

The system configuration of KBSC is shown in Figure 3. The function of each software module in KBSC is explained.

The Flowchart Editor is an interactive functional editor for creation and modification of KBSC flowcharts for design specification entry. An example of a KBSC flowchart is shown in Figure 4. Actions and conditions as well as state transitions are represented by functional macros, which are independent of hardware. These macros are used to define data and numerical operators (e.g., add, subtract, shift, logical AND, logical OR) in each state. A sample list of macros is given in Figure 5.

The Flowchart Simulator is a verification tool for edited flowcharts at the functional level. It guarantees that edited flowcharts represent correct functions. ASIC users easily can operate the Flowchart Simulator to verify functional correctness.

After simulation, flowchart information is separated into actions and conditions. Actions are used for data-path synthesis, and conditions are used for control logic synthesis. Both actions and conditions are described

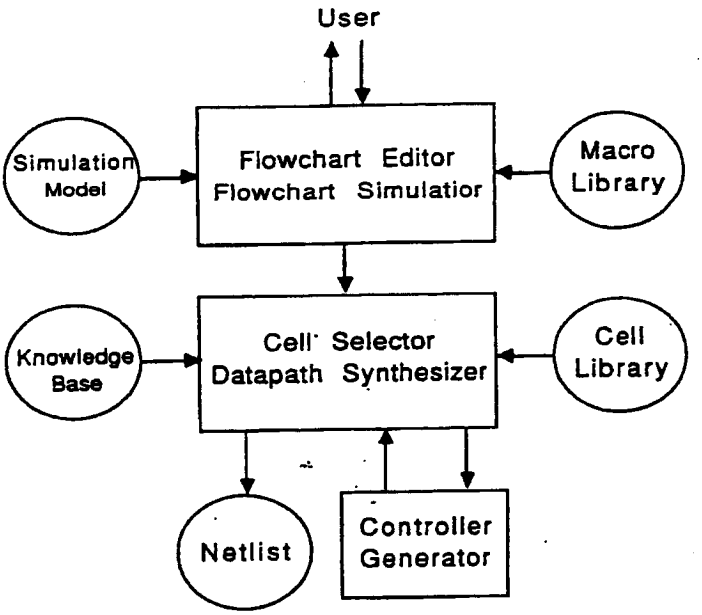


Figure 3. KBSC Configuration.

using the antecedent action form (AAF). An example of AAF is shown in Figure 6.

The Datapath Synthesizer provides automatic logic synthesis and optimization for data-path circuits. Synthesis and optimization of logic are accomplished by applying IF–THEN-type rules. Rules for placement and routing must be prepared for all macros. Example procedures for synthesis and optimization follow:

Place all macros according to rules for placement.
Place register blocks for all buses.
Route between all macros and register blocks. Add control signals to state-transition equations if needed.
Convert appropriate registers to counters and shifters. Add control signals to state-transition equations if needed.
Eliminate unnecessary macros and register blocks by classifying all macros and register blocks in terms of function and timing. Insert multiplexers into commonly used macros and register blocks, and add control signals to state-transition equations.
Connect clock signals. Clock signals for data-path circuits are synthesized separately from clock signals for control circuits. This is to assure that enable signals are stabilized before starting data transfer from data-path circuits.

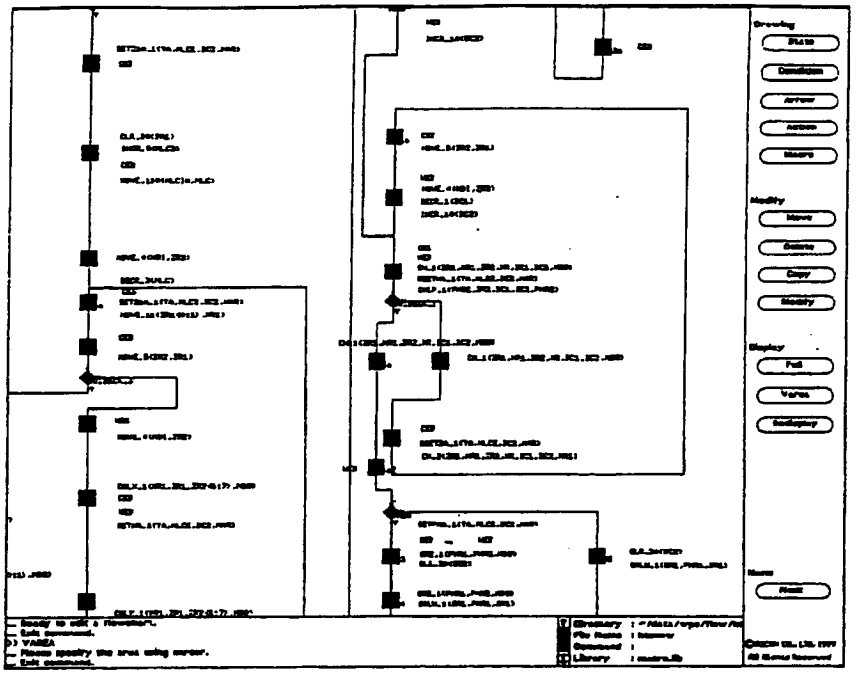


Figure 4. Example of a KBSC Flowchart.

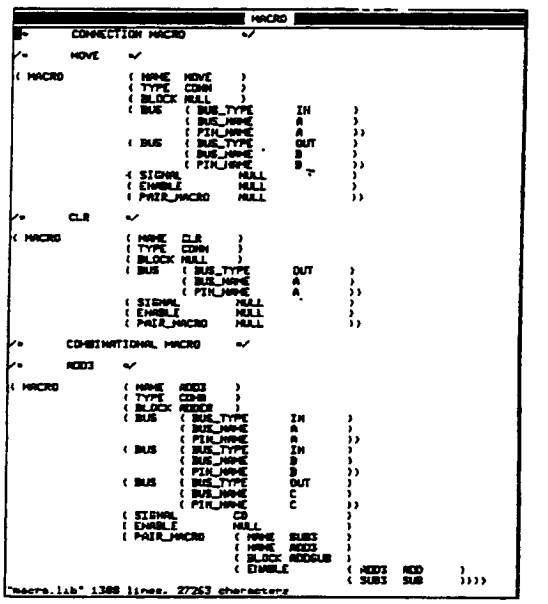


Figure 5. Sample List of Macros.

The Cell Selector uses rules to select existing cells from a library to replace functional cells (without geometrical information) used in data-path synthesis. Example procedures for cell selection follow:

Enter design constraints such as speed, power consumption, and chip area.
Select cells that satisfy entered design constraints.
Eliminate unnecessary circuit portions if a selected cell has unused terminal(s).

The Controller Generator accepts state-transition equations (including modified state-transition equations during data-path synthesis) and auto-

```
KBSC
name    htcnew;
data path        TA<0:3>,        XR<0:7>,       DC1<0:3>,
                DC2<0:3>,       HR1<0:7>,       SR1<0:5>,
                MDI<0:7>,       XR1<0:1>,       MLC<0:2>,
                ZR1<0:7>,       ZR2<0:7>,       MAR<0:7>,
                MDO<0:7>,      MLC2<0:3>,      PHR1<0:7>,
               PHR2<0:7>,     DC1IN<0:3>,     MLCIN<0:2>;
input            TA,      PHON,     DC1IN,       go,
                MDI,     MLCIN;
output           CEB,      WEB,      MDO,       MAR;
reset            x1;
{
s1 :  s2;
s1 :: WEB;
s1 :: MOVE.4(MDI,ZR2);
s1 :: DECR.1(DC1);
s1 :: INCR.10(DC2);
s2 :. BO.DECR.1 s3a;
s2 :. !BO.DECR.1 s3;
s2 :: CEB;
s2 :: WEB;
s2 :: CH.1(ZR1,HR1,ZR2,XR,DC1,DC2,MDO);
s2 :: CALP.1(PHR2,ZR2,DC1,DC2,PHR2);
s2 :: RSETHA.1(TA,MLC2,DC2,MAR);
s4 :  s1a;
s4 :: CEB;
s4 :: RSETZA.1(TA,MLC2,DC2,MAR);
s4 :: CH.2(ZR1,HR1,ZR2,XR,DC1,DC2,HR1);
x1 :. go x2;
x1 :. !go x1;
x1 :: CLR.7(MLC2);
x1 :: CLR.30(DC2);
x1 :: CLR.2(ZR1);
x2 :  x3;
x2 :: CEB;
x2 :: SETZ1A.1(TA,MLC2,DC2,MAR);
x3 :  x4;
x3 :: CEB;
x3 :: MOVE.110(MLCIN,MLC);
x3 :: INCR.5(MLC2);
x4 :  x4a;
x4 :: MOVE.4(MDI,ZR2);
x4 :: CLR.20(SR1);
x5 :. BO.DECR.3 x9;
x5 :. !BO.DECR.3 x6;
"htcnew.aaf" 129 lines, 2800 characters
```

Figure 6. Example of Antecendent Action Form.

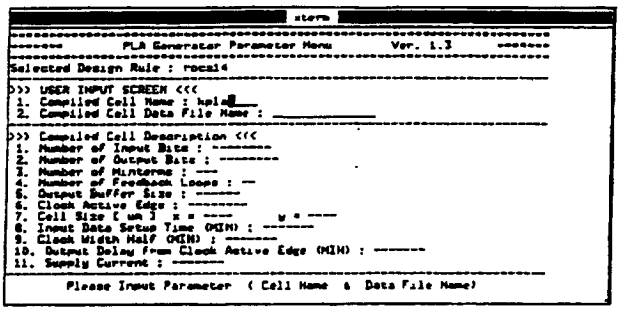


Figure 7. Parameter Menu for PLA Generation.

matically performs synthesis and optimization of logic circuits. Synthesis and optimization can be done in the following two stages.

1. Make a state assignment from state-transition equations and obtain Boolean expressions for circuits that implement state codes and output functions. Example procedures for making a state assignment follow:

   Count the total number of states and calculate the number of bits required to represent each state code.
   Assign a unique code for each state in order of appearance.
   Obtain Boolean expressions from a truth table describing present states, conditions, and next states.

2. Minimize Boolean expressions and generate a parameter menu for PLA generation (Figure 7), or generate a netlist for a logic circuit composed of existing cells. Example procedures for synthesizing a PLA-based system controller follow:

   Derive a truth table from Boolean expressions.
   Reduce the truth table by applying rules for data compression. Obtain a parameter file for PLA generation.

   Example procedures for system controller synthesis using random logic follow:

   Convert Boolean expressions to logic using only NAND gates.
   Minimize logic by applying rules for optimization.
   Convert logic to a circuit using NAND and/or NOR gates with less than or equal to six inputs each.
   Minimize logic by applying rules for optimization.
   Perform an error check for fan-out number excess and other checks.

   A logic circuit (netlist) is generated automatically by combining optimized data-path and control circuits. An example of a logic circuit generated by KBSC is shown in Figure 8.

4 RULES FOR DATA-PATH SYNTHESIS

Example rule numbers used to synthesize data-paths are shown in Table 1 in execution (firing) order. The number of execution times for each rule is also shown in Table 2. Rule descriptions for data-path synthesis follow:

Rule 10 connects clock and reset terminals to data path circuits.
Rule 100 places an input buffer for an external input bus with a corresponding data length. It also places a register for an internal bus.
Rule 200 places macros (add, sub, and, or). If the macro's first operand is a bus, then it places a block with a corresponding data length. For example, add 0.12 (OP < 16:31 > C1) places a block with 16 bits of data length.
Rule 202 places macros (not, sfr, sfl). Similar to Rule 200.
Rule 220 places a macro (cmp). Similar to Rule 200.
Rule 230 places output buffers.

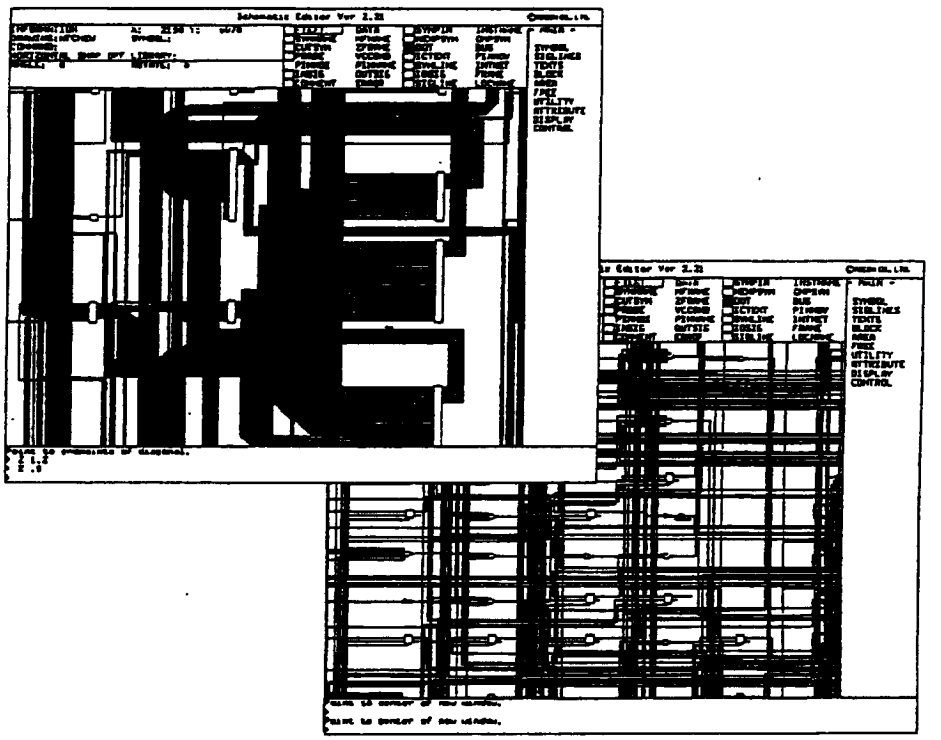


Figure 8. Example of a Logic Circuit Synthesized by KBSC.

Table 1. Rule Firing Sequence for Data-path Synthesis

CONTEXT	BEGIN	RULE 10
CONTEXT	BUSBLOCK	RULE 100
CONTEXT	MACROBLOCK	RULE 200
CONTEXT	MACROBLOCK	RULE 202
CONTEXT	MACROBLOCK	RULE 210
CONTEXT	MACROBLOCK	RULE 220
CONTEXT	MACROBLOCK	RULE 230
CONTEXT	MACROBLOCK	RULE 240
CONTEXT	CONNECTION	RULE 302
CONTEXT	CONNECTION	RULE 308
CONTEXT	CONNECTION	RULE 320
CONTEXT	CONNECTION	RULE 330
CONTEXT	CONNECTION	RULE 332
CONTEXT	CONNECTION	RULE 340
CONTEXT	CONNECTION	RULE 334
CONTEXT	CONNECTION	RULE 310
CONTEXT	CONNECTION	RULE 350
CONTEXT	CONNECTION	RULE 360
CONTEXT	CONNECTION	RULE 370
CONTEXT	CONNECTION	RULE 372
CONTEXT	CONNECTION	RULE 380
CONTEXT	TRANSFORM	RULE 400
CONTEXT	TRANSFORM	RULE 410
CONTEXT	ADDCLOCK	RULE 500
CONTEXT	ADDCLOCK	RULE 510
CONTEXT	ADDCLOCK	RULE 520
CONTEXT	SELECTION	RULE 610
CONTEXT	SELECTION	RULE 620
CONTEXT	SELECTION	RULE 630
CONTEXT	SELECTION	RULE 640
CONTEXT	SELECTION	RULE 650
CONTEXT	SELECTION	RULE 655
CONTEXT	SELECTION	RULE 660
CONTEXT	SELECTION	RULE 665
CONTEXT	SELECTION	RULE 670
CONTEXT	SELECTION	RULE 675
CONTEXT	SELECTION	RULE 680
CONTEXT	SELECTION	RULE 685
CONTEXT	SELECTION	RULE 642
CONTEXT	SELECTION	RULE 644
CONTEXT	SELECTION	RULE 600

Table 2. Number of Executions for Each Rule

RULE 10	TIME 1
RULE 100	TIME 1
RULE 200	TIME 2
RULE 202	TIME 3
RULE 210	TIME 1
RULE 220	TIME 1
RULE 230	TIME 4
RULE 240	TIME 1
RULE 300	TIME 1
RULE 302	TIME 1
RULE 308	TIME 3
RULE 310	TIME 1
RULE 320	TIME 1
RULE 330	TIME 9
RULE 332	TIME 4
RULE 334	TIME 1
RULE 340	TIME 1
RULE 350	TIME 4
RULE 360	TIME 1
RULE 370	TIME 3
RULE 372	TIME 7
RULE 380	TIME 10
RULE 400	TIME 2
RULE 410	TIME 1
RULE 500	TIME 2
RULE 510	TIME 1
RULE 520	TIME 8
RULE 600	TIME 1
RULE 610	TIME 8
RULE 620	TIME 5
RULE 630	TIME 2
RULE 640	TIME 1
RULE 642	TIME 3
RULE 644	TIME 4
RULE 650	TIME 1
RULE 655	TIME 1
RULE 660	TIME 3
RULE 665	TIME 1
RULE 670	TIME 3
RULE 675	TIME 1
RULE 680	TIME 3
RULE 685	TIME 1

Rule 240 places input buffers.
Rule 210 clears all flags to change mode from placement to routing.
Rule 300 routes macros (add, sub, and, or). It connects a bus register's output terminal, corresponding to the macro's first operand, to the macro's "A" terminal. Similarly, it applies to the macro's second and third operands.
Rule 302 is similar to Rule 300. It applies to the macro's second operand.

A total of 114 rules (42 different types of rules) are applied to synthesize data-paths automatically and to convert them to an equivalent circuit composed of existing cells.

5 RULES FOR CONTROL LOGIC SYNTHESIS

Example rule numbers used to synthesize control logic are shown in Table 3 in execution (firing) order. Rule descriptions for control logic synthesis follow:

Rule 10 constructs a truth table from Boolean expressions.
Rule 100 merges two rows in a truth table with the same combination of inputs into a single row.
Rule 500 checks the fan-out.

Table 3. Rule Firing Sequence for Control Logic Synthesis

CONTEXT	BEGIN	RULE 10	CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 450
CONTEXT	BEGIN	RULE 20	CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 480
CONTEXT	MINIMIZE	RULE 100	CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 450
CONTEXT	MINIMIZE	RULE 130	CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 450
CONTEXT	MINIMIZE	RULE 100	CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 450
CONTEXT	MINIMIZE	RULE 130	CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 450
CONTEXT	MINIMIZE	RULE 130	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	OUT	RULE 210	CONTEXT	MDLMIN	RULE 400	CONTEXT	ERRCHK	RULE 500
CONTEXT	MINIMIZE	RULE 130	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	OUT	RULE 200	CONTEXT	MDLMIN	RULE 450	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 300	CONTEXT	MDLMIN	RULE 400	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 430	CONTEXT	MDLMIN	RULE 400	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 500
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 460	CONTEXT	ERRCHK	RULE 510
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 460	CONTEXT	ADDBUF	RULE 600
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 460	CONTEXT	ADDBUF	RULE 600
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 470	CONTEXT	ADDBUF	RULE 600
CONTEXT	MDLMIN	RULE 450	CONTEXT	MDLMIN	RULE 440	CONTEXT	ADDBUF	RULE 600
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 450	CONTEXT	ADDBUF	RULE 600
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 450	CONTEXT	ADDBUF	RULE 600
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 450	CONTEXT	ADDBUF	RULE 600
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 450	CONTEXT	ADDBUF	RULE 610
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 480			
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 450			
CONTEXT	MDLMIN	RULE 480	CONTEXT	MDLMIN	RULE 450			
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 450			
CONTEXT	MDLMIN	RULE 460	CONTEXT	MDLMIN	RULE 480			
CONTEXT	MDLMIN	RULE 400	CONTEXT	MDLMIN	RULE 450			
			CONTEXT	MDLMIN	RULE 450			

A total of 110 rules (18 different types of rules) are applied to synthesize control logic automatically and to convert it to a circuit using existing cells.

6 PERFORMANCE EVALUATION

ASIC design by KBSC was compared with design by a senior engineer in terms of design time, chip performance, and gate count. An inference engine chip for real-time rule processing is used as a design example.

1. *Design time:* KBSC needed only one-fifth of the design time required by a senior engineer. KBSC took approximately three weeks to complete the chip layout, whereas a senior engineer took approximately 15 weeks. Most of the time spent by the engineer was for architecture design.
2. *Chip performance:* The number of cycles to execute an IF–THEN-type rule is defined as a unit parameter. One rule was executed with 13 cycles for the circuit designed by KBSC, whereas the circuit designed by the senior engineer took 8 cycles to execute one rule. This was mainly due to the engineer's experience in designing circuits that can process more than two pieces of data in parallel.
3. *Gate count:* The KBSC circuit required 2,800 gates; the circuit designed by the engineer required 2,500 gates. The difference of 300 gates is due to the lack of rules for eliminating unused gates and optimizing logic circuits.

7 COMPARISON BETWEEN LOGIC SYNTHESIZERS

Logic synthesis systems in References [1–8] are compared with KBSC in terms of flowchart input form and rule-based approach to automatic data-path and control logic synthesis. A technology-oriented register transfer (RT)-level description is used in [1]. Input to KBSC is not an RT-level description, as used in many other approaches. KBSC's input is also not a flowchart like specification of control for typical finite-state machine design. Rules used in KBSC do synthesize both data-path and control circuits from a hardware-independent flowchart description. Output from KBSC is a netlist for automatic placement and routing.

Input to ALERT [2, 3] is an architectural description in Iverson's APL notation with declarations of variables to represent physical devices such as flip-flops and registers. The user of ALERT needs to specify memory size, word length, instruction format, and other hardware design features. In contrast, KBSC's input is a flowchart description with functional macros that are independent of hardware.

Input to the CMU-DA system [4] is an algorithmic description in ISP language. ISP description is translated to the first-level path graph with interconnections of abstract components. It is then mapped to the second level of the data-path graph with selected physical modules.

The Design Automation Assistant (DAA) [5] is an expert system that uses heuristic rules to synthesize architectural implementation from an algorithmic description with design constraints. Input to DAA is written in ISPS, a description for a Digital Equipment Corporation computer. The DAA produces a hardware network composed of modules, ports, links, and symbolic microcode.

Input to Flamel [6] is a behavioral description in the form of a Pascal program. Flamel's input is associated with a specified bus architecture. Functional blocks such as ALUs, registers, and I/O pads are ordered and placed on buses. Multiplexers are needed to regulate bus traffic at each block's input and output. In contrast, the KBSC approach is not tied to any fixed bus architecture and needs no multiplexers to regulate bus traffic.

SOCRATES [7] uses an expert system to optimize combinatorial logic for a specific target technology. Rules used in SOCRATES optimize gate-level circuits for speed and area in a given technology. A rule replaces a portion of a circuit by a functionally equivalent but more optimal circuit.

8 CONCLUSION

It has been shown that a knowledge-based silicon compiler (KBSC) dramatically reduces time required for functional and logic design. KBSC is clearly distinguished from other logic synthesis systems in terms of its flowchart input form and rule-based approach to automatic data-path and control logic synthesis. More than 10 chips have been designed using KBSC. Applications of these chips include speech synthesis, rule processing, vector-raster conversion, Japanese-character optical character recognition, and printer control.

From these initial results, we believe that KBSC is a useful tool for designers who are not familiar with hardware or circuit design. To improve the current version of KBSC more rules are needed for parallel processing and logic optimization. Most of the problems observed can be solved by simply adding new rules to the knowledge base and modifying rules. The program is written in C language and runs on Sun and other workstations supporting the UNIX operating system and the X windows system.

REFERENCES

[1] J.A. Darringer, and W.H. Joyner, Jr., "A New Look at Logic Synthesis," *Proc. of 17th Design Automation Conf.*, pp. 543–549, 1980.

[2] T.D. Friedman, and S.C. Yang, "Methods Used in an Automatic Logic Design Generator (ALERT)," *IEEE Transactions on Computers*, Vol. C-18, No. 7, pp. 593–614, July 1969.

[3] T.D. Friedman, and S.C. Yang, "Quality of Designs from an Automatic Logic Generator (ALERT)," *Proc. of 7th Design Automation Conf.*, pp. 71–89, 1970.

[4] A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, and J. Kim, "The CMU Design Automation System—An Example of Automated Data Path

Design," *Proc. of 16th Design Automation Conf.*, Las Vegas, NV, pp. 73–80, 1979.

[5] T.J. Kowalski, D.J. Geiger, W.H. Wolf, and W. Fichter, "The VLSI Design Automation Assistant: From Algorithms to Silicon," *IEEE Design and Test of Computers Magazine*, Vol. 2, No. 4, pp. 33–43, August 1986.

[6] H. Trickey, "Flamel: A High-Level Hardware Compiler," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 2, pp. 259–269, March 1987.

[7] A.J. deGeus, and W. Cohen, "A Rule-Based System for Optimizing Combinational Logic," *IEEE Design and Test of Computers Magazine*, Vol. 2, No. 4, pp. 22–32, August 1986.

[8] L. Trevillyan, "An Overview of Logic Synthesis Systems," *Proc. of 24th ACM/IEEE Design Automation Conf.*, Paper 9.1, pp. 166–172, 1987.

[9] H. Kobayashi, "KBSC: A Knowledge Based Silicon Compiler—A Future Trend in ASIC Design," Ricoh ASIC Technical Seminar, Tokyo, Japan, October, 1987.

Hideaki Kobayashi (See guest editorial section (p. 355) of this special issue for author's biography).

Terumi Suehiro joined Ricoh Company in 1979 and has been involved in the development of CPU peripheral LSI. Since 1985 he has been developing a Ricoh CAD system for ASICs. He is currently Manager and Senior Engineer of the Semiconductors R & D Center. He received a B.S. degree in applied physics from Nagoya University in 1979.

Masahiro Shindo joined Ricoh Company in 1979 and has been involved in ASIC design, CAD tools, and strategic management. He is currently the Assistant General Manager of the Electronic Devices Division and Director of the Semiconductors R & D Center. Prior to joining Ricoh Company, he was employed by Mitsubishi Electric Corp. and was involved in the development of integrated circuit process technology (micron process, CVD) and devices (DRAM, SRAM, EPROM, LOGIC). He received a B.S. degree in industrial chemical engineering from the University of Ehime in 1963.

US005218669A

United States Patent [19]

Kobayashi et al.

[11] Patent Number: 5,218,669

[45] Date of Patent: Jun. 8, 1993

[54] **VLSI HARDWARE IMPLEMENTED RULE-BASED EXPERT SYSTEM APPARATUS AND METHOD**

[75] Inventors: **Hideaki Kobayashi**, Columbia, S.C.; **Masahiro Shindo**, Osaka, Japan

[73] Assignees: **International Chip Corporation**, Columbia, S.C.; **Ricoh Company**, Tokyo, Japan

[21] Appl. No.: **489,892**

[22] Filed: **Mar. 7, 1990**

Related U.S. Application Data

[63] Continuation-in-part of Ser. No. 166,873, Mar. 11, 1988, abandoned.

[51] **Int. Cl.**[5] **G06F 15/18**

[52] **U.S. Cl.** **395/51**; 395/52; 395/54; 395/11

[58] **Field of Search** 364/513; 395/51, 52, 395/54, 11

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,628,435	12/1986	Tashiro et al.	364/513
4,648,044	3/1987	Hardy et al.	364/513
4,649,515	3/1987	Thompson et al.	364/513
4,748,439	5/1988	Robinson et al.	364/513
4,752,890	6/1988	Natarajan et al.	364/513
4,754,410	6/1988	Leech et al.	364/513
4,783,752	11/1988	Kaplan et al.	364/513
4,815,005	3/1989	Oyanagi et al.	364/513
4,839,822	6/1989	Dormond et al.	364/513

OTHER PUBLICATIONS

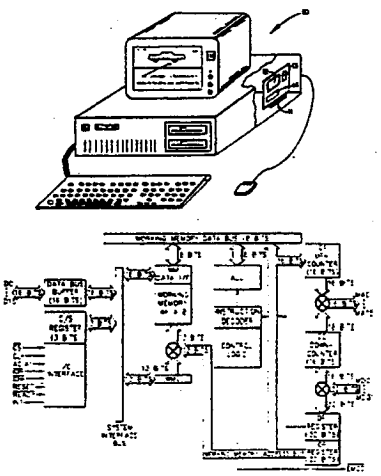
Crash Course in Artificial Intelligence and Expert Systems; Frenzel; Howard W. Sams & Co.; 1986; pp. 76–83.
Expert Systems; IEEE Potentials; Oct. 1986; Ann Miller; pp. 12–15.
An Intelligent Framework for Engineering Decision-Making; Stephen C-Y, Lu; Inter. Congress and Exposition, Detroit, MI; Feb. 1987; SAE Technical Paper Series.
AI Now More Realistically At Work in Industry; Control Engineering; Jul. 1989; Bartos; pp. 90–93.
The Use of expert systems in industrial control; Measurement and Control; vol. 17; Dec./Jan. 1984/5; Knight et al., pp. 409–413.
Robert H. Michaelson, et al., "The Technology of Expert Systems", Byte Magazine, Apr. 1985, pp. 303–312.
Beverly A. Thompson, et al., "Inside an Expert System", Byte Magazine, Apr. 1985, pp. 315–330.
Michael F. Deering, "Architectures for AI", Byte Magazine Apr. 1985, pp. 193–206.
Phillip Robinson, "The Sum: An AI Coprocessor", Byte Magazine, Jun. 1985, pp. 169–179.
Kenneth L. Short, "Microprocessors and Programmed Logic", Prentice-Hall, Inc., 1981.

Primary Examiner—Allen R. MacDonald
Attorney, Agent, or Firm—Bell, Seltzer, Park & Gibson

[57] **ABSTRACT**

The hardware-implemented rule-based expert system of this invention is suitable for performing high speed inferencing in artificial intelligence (AI) applications, and is characterized by being domain independent so that it can be applied to a variety of different application domains. The expert system includes a working memory in which, at the beginning of an inferencing operation, is stored known information or facts pertaining to the application domain. Additionally, a rule memory is provided for storing a rule set for the application domain. The rule set is comprised of a series of instructions, each defining a condition or an action. Instructions are successively loaded from the rule memory into via a first data bus. The logic unit successively executes the instructions in working memory with reference to the stored facts in working memory to thereby deduce new facts. The logic unit is coupled to working memory via a second data bus. During the inferencing operation, as new facts are deduced, they are stored in working memory and may be used for the execution of subsequent instructions. Upon the completion of the inferencing operation, an input/output interface transfers the facts stored in working memory to an output device.

47 Claims, 11 Drawing Sheets



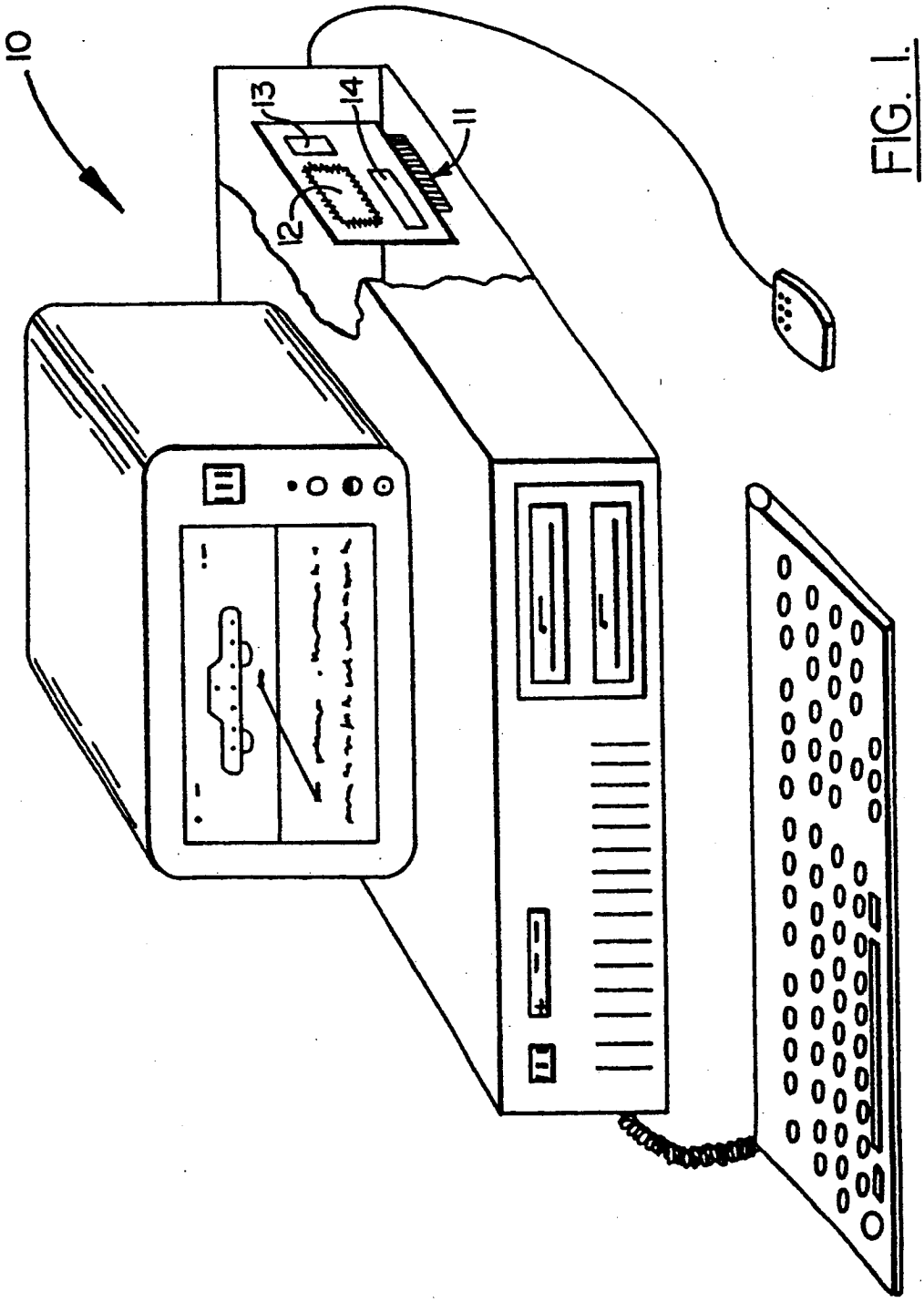


FIG. 1.

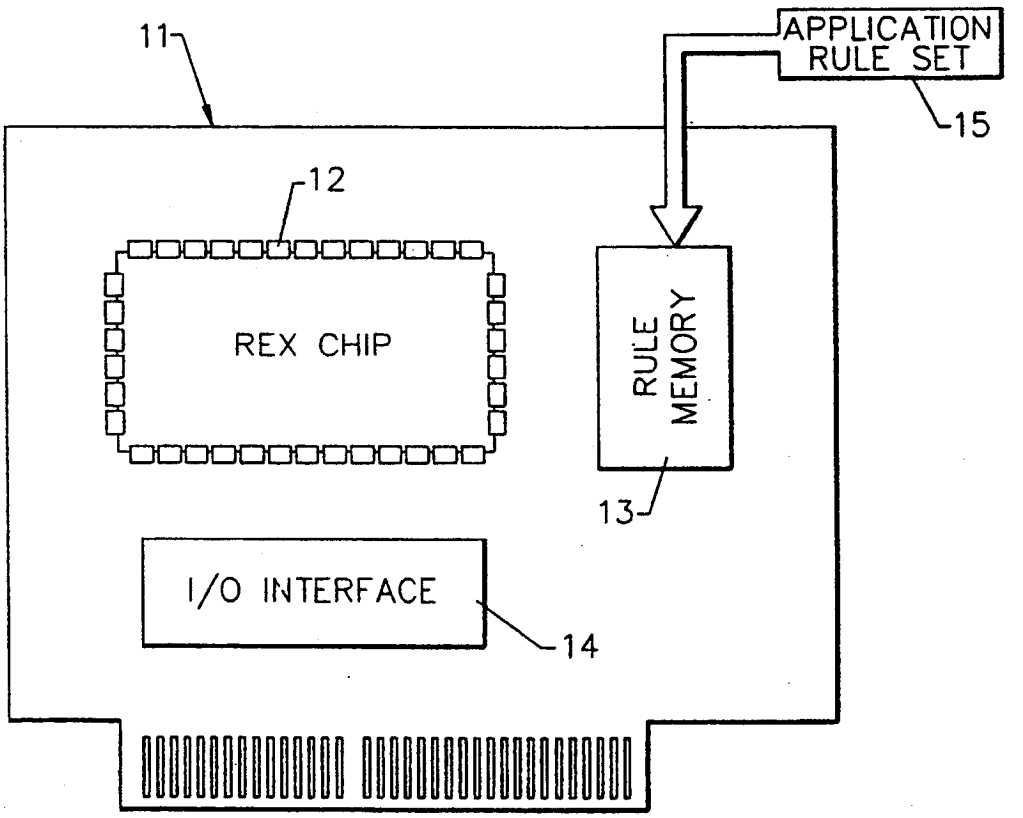

11
APPLICATION RULE SET
15
12
REX CHIP
RULE MEMORY
13
I/O INTERFACE
14

FIG. 2.

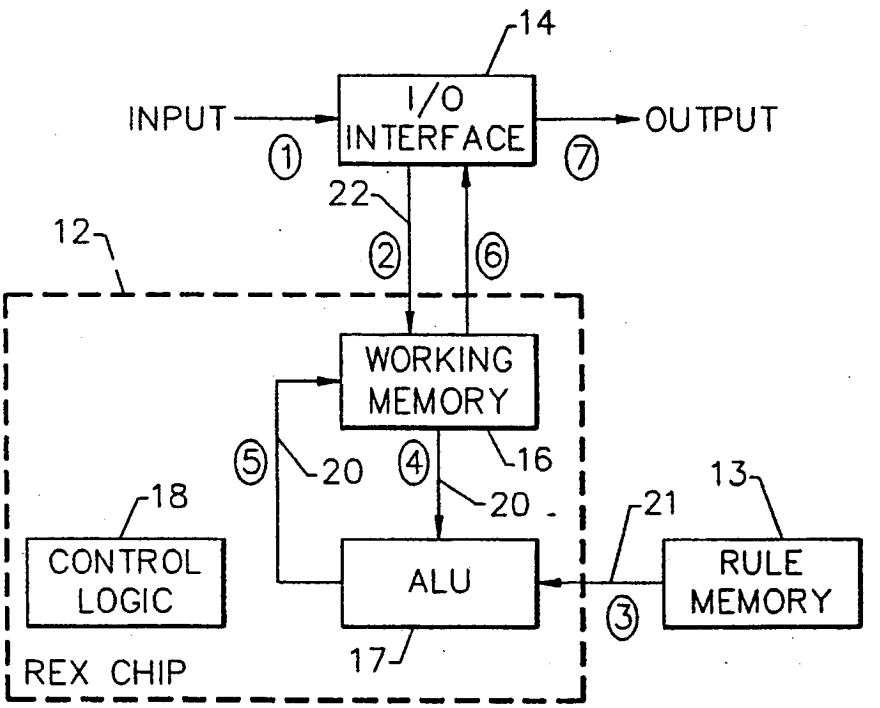

14
INPUT
I/O INTERFACE
OUTPUT
1
7
22
12
2
6
WORKING MEMORY
5
20
4
16
20
13
18
21
CONTROL LOGIC
ALU
RULE MEMORY
3
17
REX CHIP

FIG. 3.

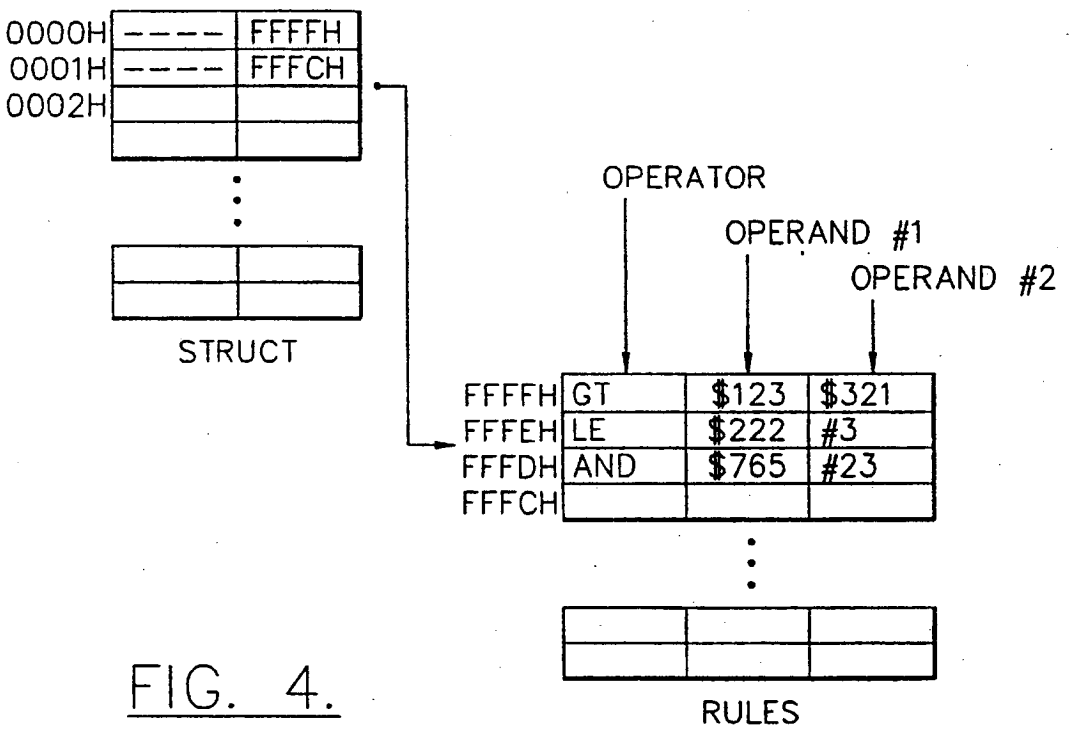

0000H
0001H
0002H
FFFFH
FFFCH
STRUCT
OPERATOR
OPERAND #1
OPERAND #2
FFFFH GT $123 $321
FFFEH LE $222 #3
FFFDH AND $765 #23
FFFCH
RULES

FIG. 4.

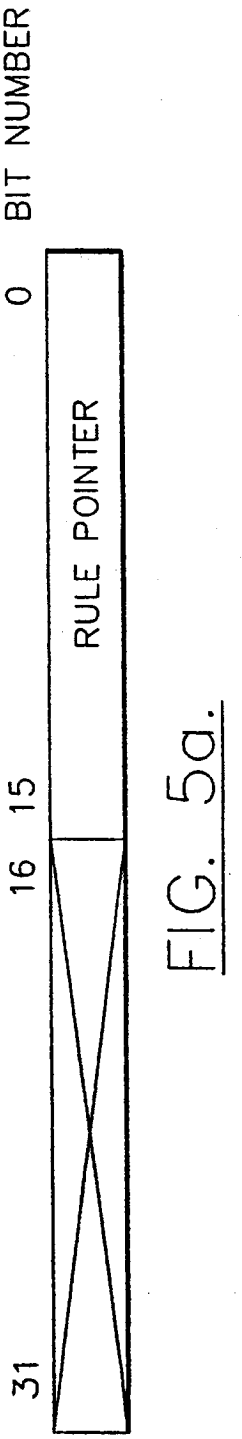

BIT NUMBER
0
16
15
31
RULE POINTER

FIG. 5a.

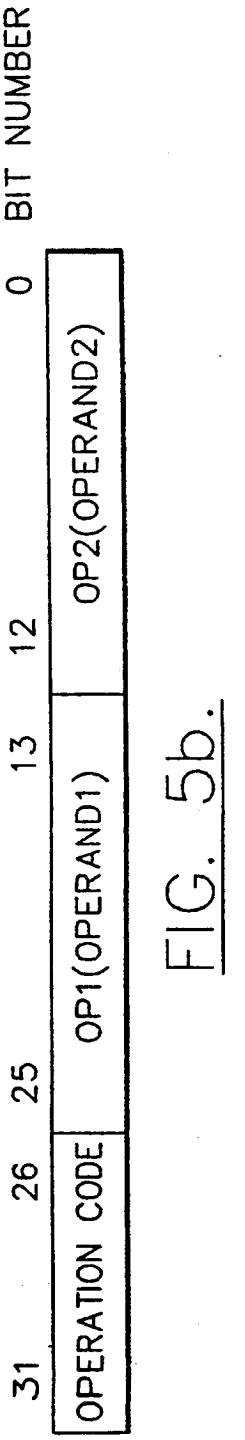

BIT NUMBER
0
12
13
25
26
31
OP2(OPERAND2)
OP1(OPERAND1)
OPERATION CODE

FIG. 5b.

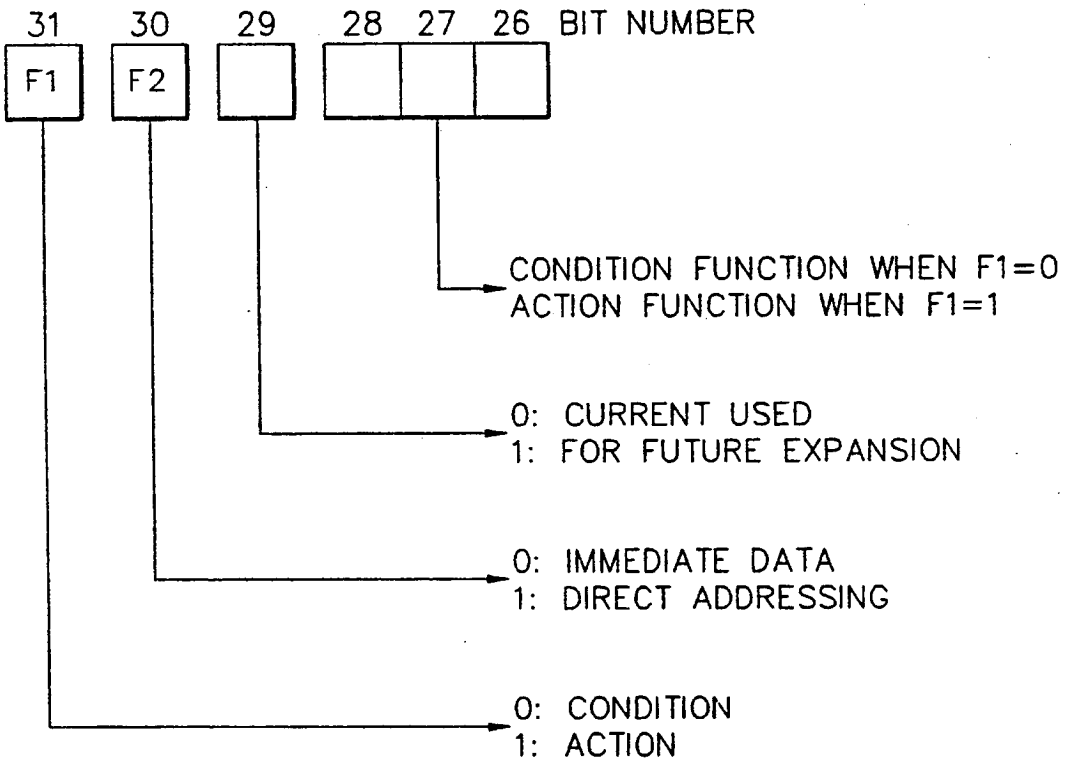

31
30
29
28
27
26
BIT NUMBER
F1
F2
CONDITION FUNCTION WHEN F1=0
ACTION FUNCTION WHEN F1=1
0: CURRENT USED
1: FOR FUTURE EXPANSION
0: IMMEDIATE DATA
1: DIRECT ADDRESSING
0: CONDITION
1: ACTION

FIG. 6.

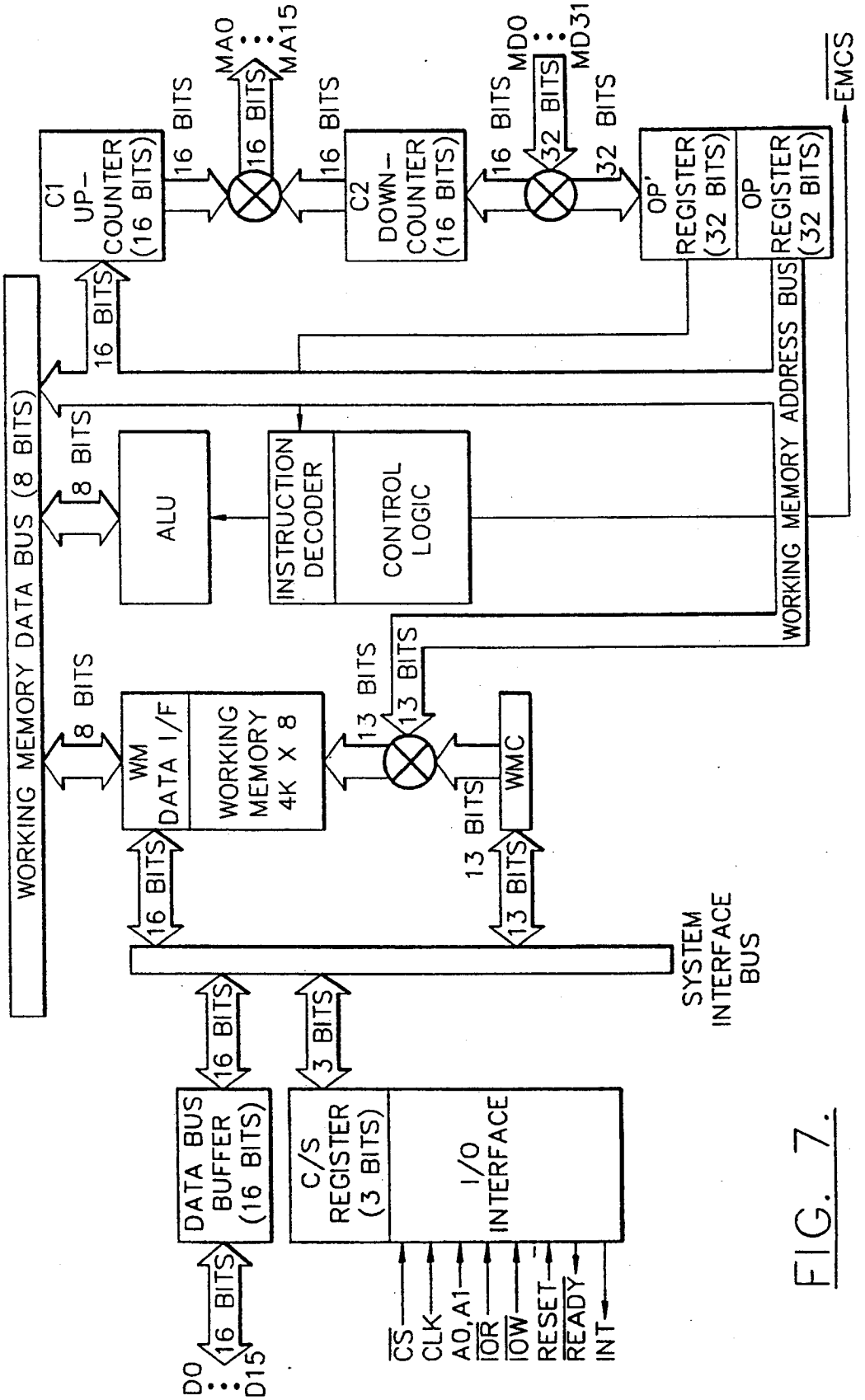

C1 UP-COUNTER (16 BITS)
16 BITS
MA0 ... MA15
16 BITS
16 BITS
C2 DOWN-COUNTER (16 BITS)
16 BITS
MD0 ... MD31
32 BITS
32 BITS
OP' REGISTER (32 BITS)
OP REGISTER (32 BITS)
EMCS
16 BITS
WORKING MEMORY DATA BUS (8 BITS)
8 BITS
ALU
INSTRUCTION DECODER
CONTROL LOGIC
WORKING MEMORY ADDRESS BUS
8 BITS
WM DATA I/F
WORKING MEMORY 4K X 8
13 BITS
13 BITS
WMC
13 BITS
16 BITS
13 BITS
SYSTEM INTERFACE BUS
16 BITS
3 BITS
DATA BUS BUFFER (16 BITS)
C/S REGISTER (3 BITS)
I/O INTERFACE
16 BITS
D0 ... D15
CS
CLK
A0, A1
IOR
IOW
RESET
READY
INT

FIG. 7.

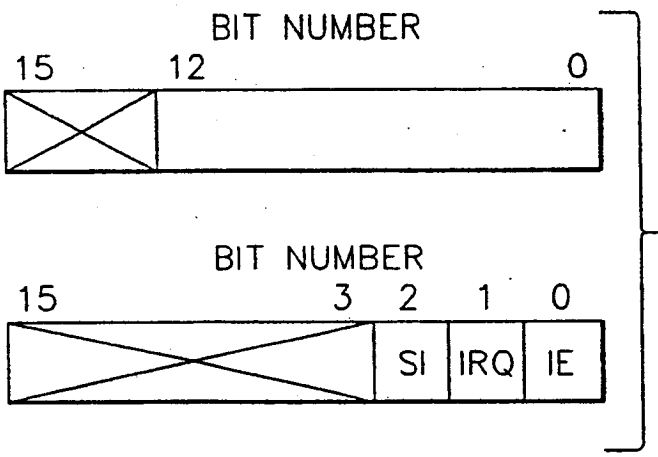


FIG. 8.

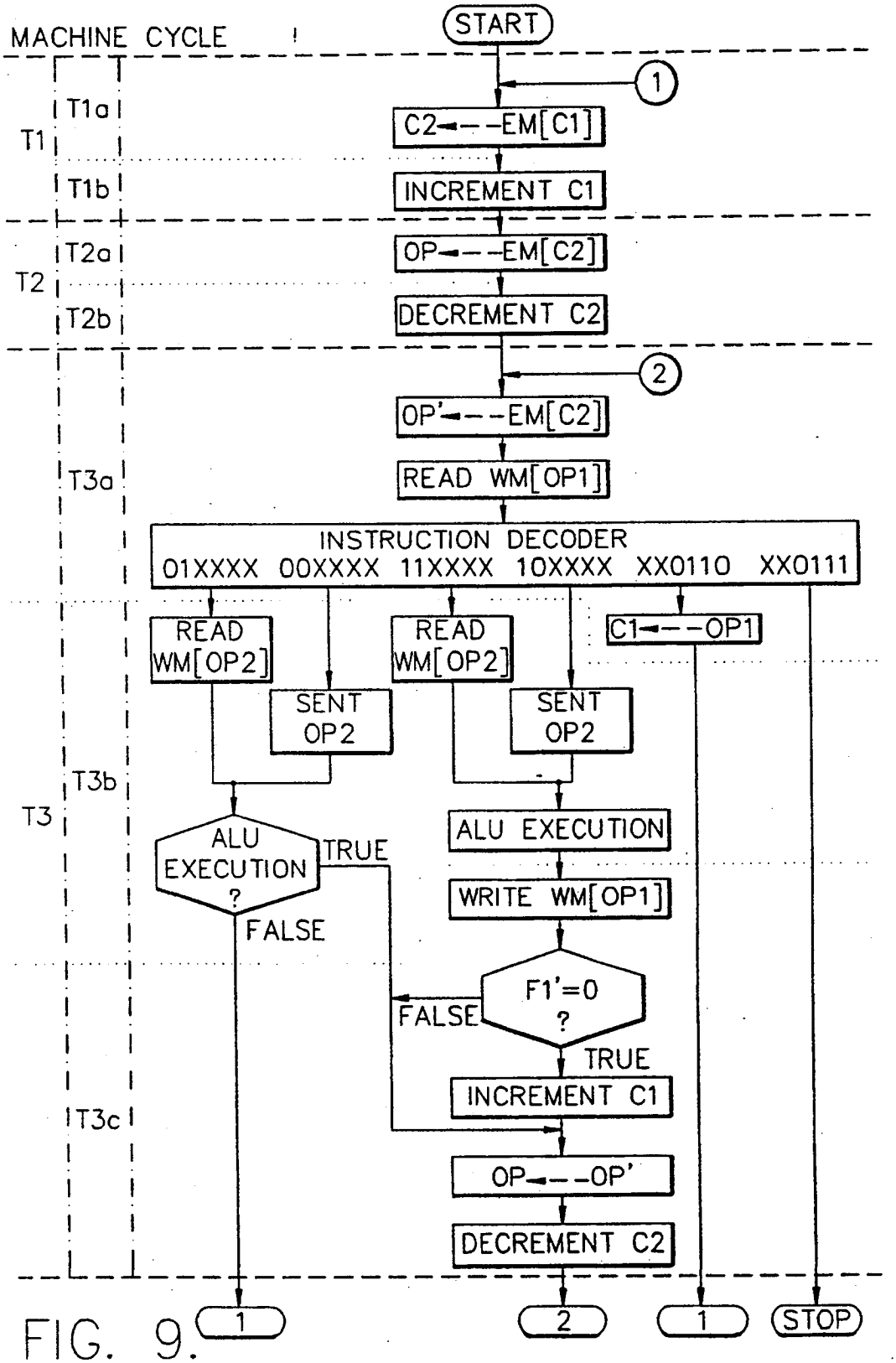

MACHINE CYCLE
START
1
T1a
T1
C2←EM[C1]
T1b
INCREMENT C1
T2a
T2
OP←EM[C2]
T2b
DECREMENT C2
2
OP'←EM[C2]
T3a
READ WM[OP1]
INSTRUCTION DECODER
01XXXX 00XXXX 11XXXX 10XXXX XX0110 XX0111
READ WM[OP2]
READ WM[OP2]
C1←OP1
SENT OP2
SENT OP2
T3b
T3
ALU EXECUTION ?
TRUE
FALSE
ALU EXECUTION
WRITE WM[OP1]
F1'=0 ?
FALSE
TRUE
INCREMENT C1
T3c
OP←OP'
DECREMENT C2
1
2
1
STOP

FIG. 9.

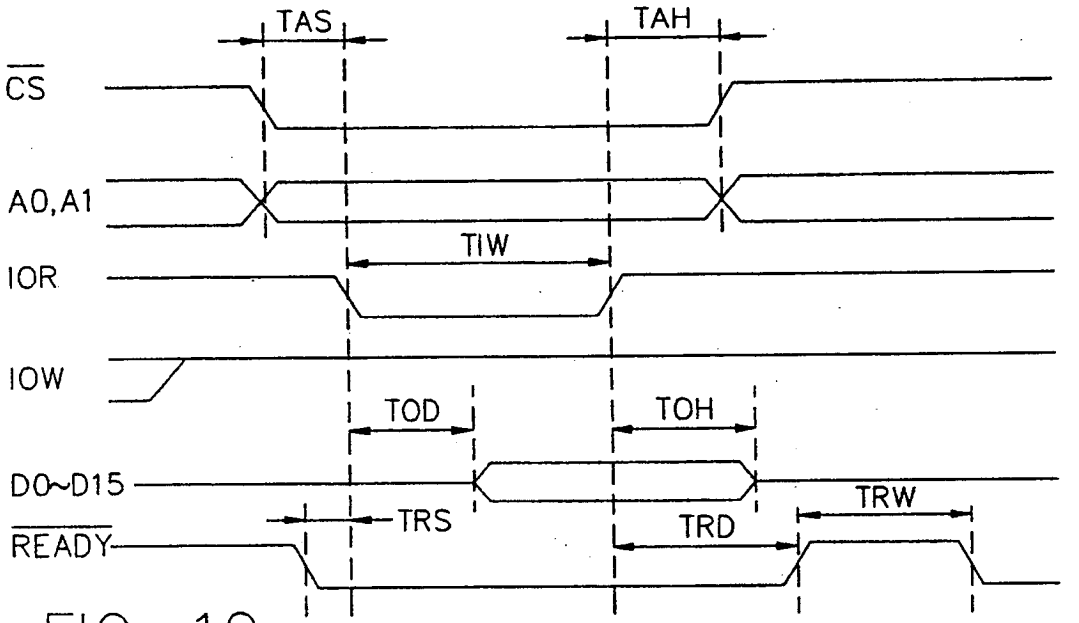

TAS
TAH
CS
A0,A1
TIW
IOR
IOW
TOD
TOH
D0~D15
TRW
TRS
TRD
READY

FIG. 10.

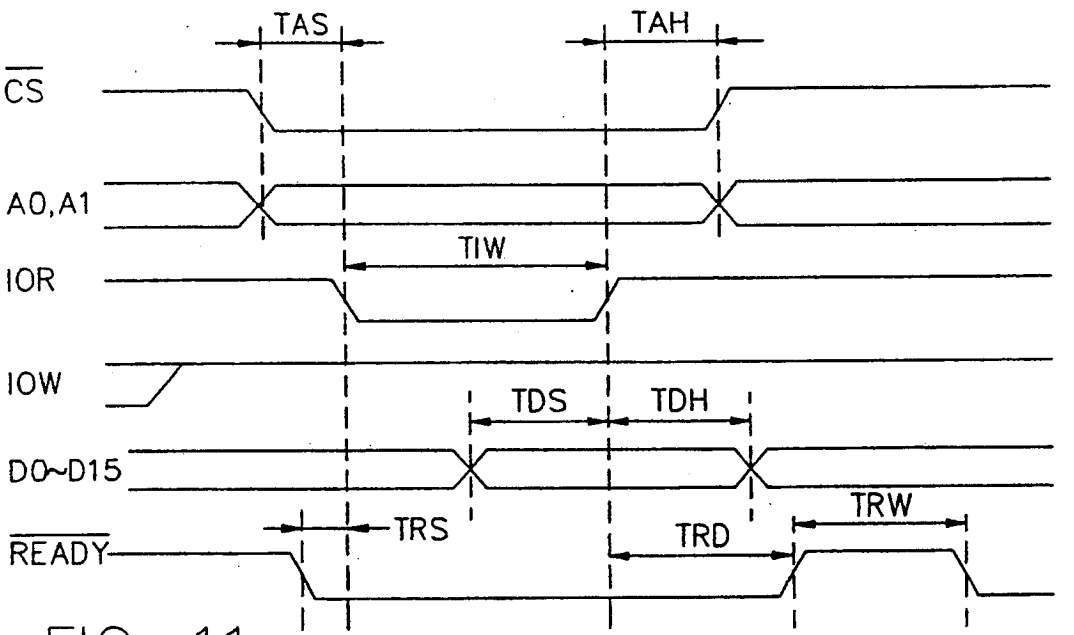

TAS
TAH
CS
A0,A1
TIW
IOR
IOW
TDS
TDH
D0~D15
TRW
TRS
TRD
READY

FIG. 11.

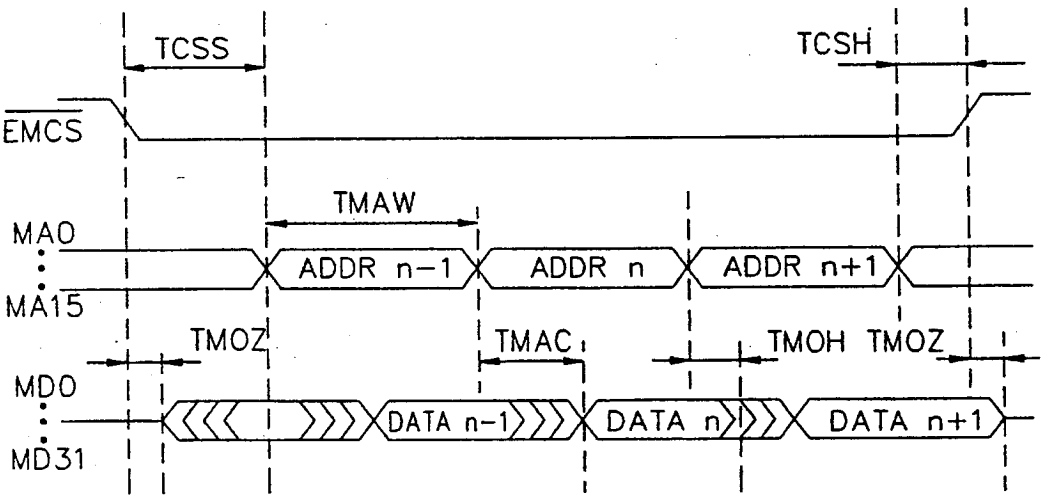

TCSS
TCSH
EMCS
TMAW
MA0
MA15
ADDR n−1
ADDR n
ADDR n+1
TMOZ
TMAC
TMOH
TMOZ
MD0
MD31
DATA n−1
DATA n
DATA n+1

FIG. 12.

VLSI HARDWARE IMPLEMENTED RULE-BASED EXPERT SYSTEM APPARATUS AND METHOD

CROSS REFERENCE TO RELATED APPLICATION

This application is a Continuation-in-Part of co-pending application Ser. No. 07/166,873 filed Mar. 11, 1988 abandoned.

FIELD AND BACKGROUND OF THE INVENTION

This invention relates to a rule-based expert system, and more particularly to a hardware implemented rule-based expert system suitable for performing high speed inferencing in artificial intelligence (AI) applications.

Expert systems are a class of computer programs that incorporate artificial intelligence (AI) technology to address problems normally thought to require human experts or specialists. In a rule-based expert system, expert knowledge in a particular application domain is represented in the form of a series of rules, "production rules". In the operation of a typical expert system the user, through a convenient user interface, supplies the expert system with certain known information about a particular problem, and the expert system applies the production rules to this known information to deduce facts and solve problems pertaining to the application domain. For further background information pertaining to expert systems, reference is made to the following articles: Robert H. Michaelsen, et al., "The Technology of Expert Systems", *Byte Magazine*, April 1985, pp. 308–312; Beverly A. Thompson, et al., "Inside an Expert System", *Byte Magazine*, April 1985, pp. 315–330; Michael F. Deering, "Architectures for AI", *Byte Magazine*, April 1985, pp. 193–206.

Successful expert systems have been developed for a number of application domains: for making medical diagnoses, for identifying organic compounds, for selecting oil drilling muds, and so forth. Additionally, a number of domain-independent expert system shells in software form have been developed to facilitate building rule-based expert systems for specific application domains. Several commercially available expert system tools are described in the above-noted articles. Typically, these expert systems and expert system tools are in the form of software programs designed to run on a general purpose computer or microcomputer. The software program provides an interactive session between the user and the expert system in which the expert system asks questions of the user and employs its expert knowledge base to solve problems and provide advice to the user.

There has been considerable interest in expanding the use of expert systems into other practical applications, and especially in developing expert systems capable of use in real-time applications. Such systems would be useful, for example as a control system for various applications, such as manufacturing processes, process control, guidance systems, robotics, etc. However, a major limitation in the development of complex, real-time AI systems is the speed of computation. In order to make effective practical use of artificial intelligence technology the efficiency and speed of computation must be increased.

Significant efforts have been made to improve the speed of AI processing by improving and streamlining the software tools used in AI processing, such as the AI languages. It has also been recognized that performance improvements can be achieved by custom designed hardware specifically engineered for artificial intelligence processing. As indicated in the Deering article noted above, one approach with respect to hardware architectural improvements has involved refinements in the processor's instruction set to allow the processor to operate more quickly. Attention has also been given to developing parallel processing architectures which would allow the AI computations to be carried out in parallelism. The article also notes that custom VLSI hardware could be employed to accelerate particular operations such as matching and fetching, parallel-processor communication and signal-to-symbol processing.

SUMMARY OF THE INVENTION

An important object of the present invention is to improve the speed and efficiency of rule-based expert systems by providing an inference engine which is implemented in hardware. More particularly, the present invention provides an application specific integrated circuit designed especially to perform high speed inferencing for a rule-based expert system.

The hardware-implemented rule-based expert system apparatus and method of the present invention is referred to herein by the acronym REX (Rule-based Expert), and includes a working memory in which, at the beginning of an inferencing operation, is stored known information or facts pertaining to the application domain. Additionally, the apparatus includes a rule memory for storing a rule set for the application domain. The working memory and rule memory are communicatively connected to the inference engine via physically separate first and second communications (data) busses, respectively. The rule set is comprised of a series of instructions, each defining a condition or an action. Means is provided for loading from the rule memory into the inference engine memory successive instructions of the rule set via the second communications bus. A logic means is provided in the inference engine for successively executing the instructions with reference to the stored facts in working memory obtained via the first communications bus. New facts are thereby deduced at high speed. During the inferencing operation, as new facts are deduced, they are stored in working memory via the first communications bus, and may be used during the execution of subsequent instructions of the rule set to derive additional facts. Upon the completion of the inferencing operation, the facts stored in working memory are transferred to an output device.

Each of the instructions of the rule set includes an operator, a condition/action flag, and a pair of operands. The logic means includes an instruction decoder for testing the condition/action flag of each instruction to determine whether the instruction is a condition or an action. If the instruction is a condition, the operands are compared in accordance with the logical operation specified by the operator to generate a logic result (e.g. true or false). If the instruction is an action, the action specified by the operator is performed on the operands.

The working memory and the logic means may be suitably provided in an integrated circuit. The rule memory may either be external to the integrated circuit and connected to the logic means via a suitable external memory bus, or the rule memory may also be provided on the integrated circuit with appropriate data bus in-

terconnections with the logic means. In either case, separate busses are provided for connecting the working memory to the logic means and the rule memory to the logic means. Since the application rule set is stored in a memory device, the REX inference engine is domain independent and can be used in any number of different applications simply by installing a different application rule set in the rule memory. The structure of the rule memory and the data structure of the application rule set are designed to greatly enhance the efficiency of the inferencing process.

To illustrate the versatility and general wide applicability of the present invention, the detailed description which follows shows how the REX inference engine can be used as co-processor in conjunction with an existing computer or microcomputer to provide an expert system capable of performing inferencing at rates significantly greater than that which could be performed by conventional software implemented inference engines. The REX inference engine can also be utilized however, in many other applications, such as a stand-alone system when provided with an appropriate internal control system, user interface, and input/output devices.

The REX inference engine is capable of performing real-time knowledge processing based upon current VLSI technology. The speed of problem solving is measured by logical inferences per second (LIPS) instead of floating point operations per second (FLOPS). One LIPS is equivalent to approximately 100 to 1000 FLOPS on a conventional computer.

BRIEF DESCRIPTION OF THE DRAWINGS

Some of the features and advantages of the invention having been stated, others will become apparent from the detailed description which follows, and from the accompanying drawings, in which

FIG. **1** is a perspective view illustrating how the REX inference engine of the present invention may be utilized as a co-processor in a conventional personal computer;

FIG. **2** is a more detailed view of a co-processor board employing the REX inference engine;

FIG. **3** is a block schematic diagram showing the data flow for the REX engine;

FIG. **4** is a block diagram illustrating the rule base structure for the REX engine;

FIG. **5***a* and **5***b* are is a diagram showing the data structure of the instructions which are stored in rule memory;

FIG. **6** is a diagram illustrating the operation codes format used in the REX chip;

FIG. **7** is an overall block diagram of the major functional components of the REX chip;

FIG. **8** is a diagram illustrating the data bus bit assignment for I/O read/write operations;

FIG. **9** is a flowchart showing the inferencing flow of the REX chip; and

FIGS. **10** to **12** are timing charts for the REX chip showing the timing of the read mode, write mode, and external memory respectively.

DETAILED DESCRIPTION

The present invention now will be described more fully hereinafter with reference to the accompanying drawings, in which an illustrative embodiment of the invention is shown. This invention can, however, be embodied in many different forms and should not be construed as limited to the embodiment set forth herein; rather, applicant provides this embodiment so that this disclosure will be thorough and complete, and will fully convey the scope of the invention to those skilled in the art.

Referring now more particularly to the drawings, FIG. **1** illustrates an expert system in accordance with the present invention which is designed to operate on a microcomputer **10**, such as an IBM AT Personal Computer with an added Rule-based Expert system (REX) co-processor board **11**. The REX board **11** consists of a REX chip **12**, external rule memory **13** and an I/O interface **14**. The REX board is illustrated in FIG. **2** in greater detail. An application rule set for a particular application domain, indicated at **15** in FIG. **2**, is stored in external rule memory **13**. Thus, the REX chip **12** is domain independent and can be utilized in a variety of different applications.

Referring to FIG. **2**, each component of the REX co-processor board **11** is explained as follows:

I/O Interface: The I/O interface **14** is responsible for the communication between the personal computer **10** and the REX co-processor board **11**. External data is transferred from the personal computer **10** to the REX board **11** via the I/O interface **14**. In the preferred embodiment illustrated herein, a DMA channel provides a communication link between the REX board **11** and the personal computer **10**. A software program run by the personal computer is employed to provide an easily understandable user interface.

REX Chip: The REX chip **12** is a hardware inference engine and forms the heart of the REX co-processor board **11**. Two major components of the REX chip are working memory and control logic. Before the inferencing process is begun, the working memory is initialized with external data from the I/O interface. External data pertaining to facts which are known about the application domain are stored in particular memory locations of the working memory. During the inferencing process, the working memory is a temporary storage for intermediate data. When the inferencing process is completed, the working memory contains the results of the inferencing process, which is then transferred to the personal computer via the I/O interface.

Rule Memory: The knowledge engineer extracts a set of production rules, called an application rule set **15**, from the application domain and this rule set is stored in the rule memory **13**. During the inferencing process, the REX chip **12** refers to rule memory **13** for rule information. The structure of the rule memory is well designed to match REX chip requirements and to reduce memory space. The data structure of the application rule set stored in rule memory is designed to greatly enhance the efficiency of the inferencing process. Further details about the structure of the rule memory and the application rule set stored therein are provided hereinafter.

The rule memory can be a ROM, RAM, EPROM, or other suitable memory device. If a RAM is used for rule memory, an initialization program is utilized to initially install the application rule set **15** in the external memory **13**.

While the specific embodiment illustrated herein demonstrates how the REX chip **12** can be utilized as a co-processor for a personal computer, persons skilled in the art will recognize that the hardware-implemented rule-based expert system of the present invention (REX) can be utilized in many other specific applica-

tions. For example, it can be utilized as a stand-alone system. In such event, a control system is provided to handle user interface and I/O interface, and additional I/O devices such as a keyboard, graphics display, etc. are provided to permit communication between the REX board and the user.

Inferencing Mechanism

There are several types of inferencing methods that can be used to solve a problem in a rule-based expert system. Some of the major inference methods are forward chaining, backward chaining, and combination chaining. The inference engine specifically illustrated and described herein uses the forward chaining inferencing method or the backward chaining inferencing method with production rules. However, it will be understood by those having skill in the art that combination chaining and other inferencing methods, now known or developed in the future, may also be used.

The rules of the rule-based system are represented by production rules. The production rule consists of an *if* part and a *then* part. The *if* part is a list of one or more conditions or antecedents. The *then* part is a list of actions or consequents. Thus, a production rule can be represented as follows:

```
if      condition_1,
        condition_2,
          .
          .
          .
        condition_n
then    action_1,
        action_2,
          .
          .
          .
        action_n
```

If the conditions {condition_1, condition_2, . . . condition_n} satisfied by the facts of a given problem, we can say that the rule is triggered. The expert system can then execute the given actions. Once the actions are executed then the rules are said to be fired. These particular actions may change other conditions, which may in turn fire other rules. The flow of rules firing will continue until the problems are solved, or no other rules can be satisfied. This method of rule firing is moving forward through the rules, hence we call this forward chaining. Forward chaining is also referred to as a deduction system or facts driven because the facts guide the flow of the rules being fired.

The triggering of the rules does not mean that the rules are fired, because the conditions of several other rules may be satisfied simultaneously, and all being triggered. Should this happen, it is necessary to apply a conflict resolution strategy to decide which rule is actually fired. A conflict resolution strategy is a process of selecting the most favorable rule where more than one rule is satisfied. Examples of conflict resolution strategies are the following:

1. The rule containing the most recent data is selected. This strategy is called Data Recency Ordering.

2. The rule which has the most complex of the toughest requirements is selected. This is also called Context Limiting Strategy.

3. The rule declared first in the list is selected. This is called Rule Ordering.

EXAMPLE 1

Forward Chaining Example

This example provides a general illustration of the operation of a rule-based expert system. For this illustration, refer to the Animal Identification Rule Set in Appendix A. This rule set tries to identify an animal by giving its physical characteristics. Assume that the following characteristics have been observed:

the animal has hair,
the animal eats meat,
the animal has a tawny color,
the animal has black stripes.

These observations are translated into the following facts:

covering=hair,
food=meat,
color=tawny,
stripes=black.

Given these facts, RULE 1 is triggered. According to the rule, we deduce that

class=mammal

Now the system takes this as a new fact, that the animal is a mammal. Hence RULE 2 through RULE 4 cannot be triggered. The condition of RULE 5 is valid, thus the system will deduce that the animal is a carnivore.

carnivore=yes

So far, the system has deduced two new facts that can be used. The first three conditions of RULE 9 are true, but the last condition is not, thus RULE 9 failed. RULE 10 is triggered and can be fired. The system thus deduces that the animal is a tiger.

animal=tiger

The inferencing does not stop here, because there are more rules. In this case none of the other rules can be satisfied. The system identifies that the animal is a tiger.

The example shows the inferencing method by working forward from the current situation of facts or observations toward a conclusion.

EXAMPLE 2

Backward Chaining Example

This Example provides a general illustration of the operation of backward chaining in a rule based expert system. Because Rule Sets can be defined as a sequence of IF-THEN statements, one approach to solving Backward Chaining problems is to transform the Rule Set into a Forward Chaining structure. As described above, REX processes the IF component of an individual rule first and if all conditions are satisfied, all consequent actions are performed. In Backward Chaining problems, a goal (final) state is identified, and then all supporting rules are examined to see if the goal can be realized. This process involves testing if the action part of one rule is related to the condition part of another rule. When this process cannot proceed any further, the inferencing process pauses, asks for any missing information, and proceeds to prove or disprove the assumption (goal).

To implement this behavior, the original Rule Set must be transformed into a Backward Chaining version of the Rule Set. This process may occur during Rule Set compilation and requires the examination of rules to rearrange and reformat them into a backward representation.

The following Example 2 illustrates how a simple Forward Chaining problem is transformed into an IF-

THEN expression of the corresponding Backward Chaining problem. The Rule Set is assumed to comprise the following rules:

Rule 1:	IF A and B THEN D
Rule 2:	IF B THEN F
Rule 3:	IF A and C THEN E
Rule 4:	IF D and C THEN J
Rule 5:	IF D THEN H
Rule 6:	IF E THEN H
Rule 7:	IF F THEN G
Rule 8:	IF E and F THEN J

G, H and J are goals or conclusions to be determined by the expert system. Accordingly, for Backward Chaining, the goals of G, H and J are identified. The goals are successively assumed to be TRUE (indicated by a "prime" symbol such as G') and the supporting rules are examined to determine which facts are necessary for the goal to be TRUE. If the facts are present, then it is assumed the goal is TRUE. If not the goal is FALSE. For example, the following inferencing sequence occurs for G:

Assume G':	
IF G' THEN F'	(Rule 7)
IF F' THEN ask (B)	(Rule 2)
IF B THEN G.	

In other words, if B is TRUE then the assumption made about G was correct.

The following inferencing sequences occur for H:

Assume H':	
(1) IF H' THEN E'	(Rule 6)
IF E' THEN ask (A), ask (C)	(Rule 3)
IF A and C THEN H.	
(2) IF H' THEN D'	(Rule 5)
IF D' THEN ask (A), ask (B)	(Rule 1)
IF A and B THEN H.	

The two Backward Chaining inferencing sequence will produce the goal of H if either A and B is TRUE or A and C is TRUE.

The following inferencing sequences occur for J:

Assume J'	
(1) IF J' THEN D', ask (C)	(Rule 4)
IF D' THEN ask (A), ask (B)	(Rule 1)
IF A and B and C THEN J.	
(2) IF J' THEN E' and F'	(Rule 8)
IF E' THEN ask (A), ask (C)	(Rule 3)
IF A and C THEN E	(Rule 3)
IF F' THEN ask (B)	(Rule 2)
IF B THEN J.	

The two Backward Chaining sequences will produce the goal of J if A and B and C is TRUE.

It will be understood by those having skill in the art that a combination of forward and backward chaining may also be used.

REX Inference Engine Architecture

The major components of the REX inference engine are shown in greater detail in FIG. 3. The REX chip itself has three primary functional components: the working memory 16, an arithmetic logic unit (ALU) 17 and control logic 18. A first data bus 20 is provided for bidirectional communication between working memory 16 and ALU 17. In the embodiment illustrated herein, the rule memory 13 is a separate memory device connected to the ALU 17 by a second data bus 21. However, those skilled in the art will recognize that the rule memory could, if desired, be integrated into the REX chip itself provided that a data bus 21 connecting rule memory 13 and ALU 17 is provided, with data bus 21 being physically distinct from data bus 20. The I/O interface 14 is communicatively connected to the working memory by a system interface bus, generally indicated at 22. The control logic is schematically represented in FIG. 3 and indicated by the reference character 18. In general, the function of the control logic 18 is to control the operations of the other elements, such as the ALU 17 and working memory 16.

Data Flow in the REX Inference Engine

The flow of data for the REX engine will be best understood from FIG. 3 and the description which follows. The circled numbers in FIG. 3 correspond to the following numbered topic headings:

1. Input Data

The user inputs the facts to the system through a user interface program on the personal computer 10. The user presents the facts in a predefined syntax. For instance, using the factual data of the Example 1 and the Rule Set of Appendix A, the user would enter the following:

```
covering = hair
color = tawny
...
... etc.
```

The user interface program converts each factual observation into a values represented by a pair of binary numbers. The first part of the pair is an address and the second part of the pair is a value.

Address	Value

In the above example we have

```
(address $32)covering = (value#10)hair,
(address $58)color = (value #55)tawny.
...
... etc.
```

where, "$" and "#" indicate that the number referred to is an address and a value, respectively. In the above case covering is mapped to address 32 (no other word maps address 32). Thus each word is assigned to a unique address number. The value hair is stored in address 32. These numbers are used in Step 2.

2. Store Facts into Working Memory

In Step 2 the facts are stored in Working Memory.

3. Fetch Rules into ALU

External Rule Memory 13 is used to store rules pertinent to the application domain. Each rule is represented as follows:

```
IF
        condition 1 and
        condition 2 and
        ...
THEN
        action 1
        action 2
        ...
```

A condition element is
(class=mammal)
Similarly an action element is:
(type=ungulate)
Each element, whether condition or action part of the rule, is represented internally as an instruction in the format shown below:

Operand1	Operand2	Operator	Dir/Imme	Act/Cond

Each instruction is of a predetermined length, for example 32 bits. Operand1 represents an address of Working Memory. Depending on the value of Dir/Imme field, Operand2 is either an address or a value in the Working Memory. Dir/Imme field specifies whether the addressing mode of Operand2 is Direct or Immediate. The Act/Cond field specifies whether the element refers to condition or action part of a rule. The Operator field specifies the type of operator used in the condition part of the rule. Example operators are : equal to (=), greater than (>), less than (<), etc.

4–5 Inferencing Cycle

The following steps are executed during the inferencing cycle.

4.1 Fetch External Memory Element

A rule is fetched from Rule Memory 13 and the Cond/Act field of the first instruction of the rule is examined to check if it is a condition or an action. If the instruction is a condition element, then the procedure described in Section 4.1.1 is used. If it is an action, then the procedure described in Section 4.1.2 is used.

4.1.1 Matching Rule Condition Element to Working Memory

The address in Operand1 is loaded into ALU (Step 4). Next the Dir/Imme field is checked to see if Operand2 is Direct or Immediate. If it is immediate, then the value of Operand2 is directly input to ALU, otherwise the contents of the address pointed by Operand2 is input to ALU. The inputs to ALU are compared by the ALU using the operator (Operator field) to determine whether the condition is true or false. If the condition is true, the next successive instruction of the rule is examined by repeating the sequence of steps indicated in section 4.1. If the condition element is false, then this rule is discarded and the next rule is tested by repeating the sequence of steps in Section 4.1.

4.1.2. Action Part

The Dir/Imme flag of the action element is first checked. If it is Direct, then the value stored at Working Memory location Operand2 is copied to the Working Memory address represented by Operand1. If Dir/Imme flag is Immediate, then Operand2 is copied to the Working Memory address represented by Operand1. After performing the action defined by the instruction, the next successive action instruction of the rule is read and the procedure described in Section 4.1.2 is repeated. If action instruction is the last instruction of the rule then, next rule is tested by repeating the sequence of steps in Section 4.1.

6. Facts to Data

After all the rules have been processed, the control is transferred to the I/O interface 14. The numerical representation of the facts is translated to a form which will be readily understood to the user.

7. Data Output

The I/O interface will then output the data to the personal computer 10.

EXAMPLE 3

REX Data Flow Example

This example illustrates how the REX chip solves the problem described above in Example 1. Again, the numbered topic headings correspond to the circled numbers in FIG. 3. Refer to Appendix A for the complete Animal Identification Rule Set.

1. Input External Data

The data or observation made is:
the animal has hair,
the animal eats meat,
the animal has a tawny color,
the animal has black stripes.
The above data enters I/O interface and is translated into facts. The data is translated into the following facts:
(address $32) covering=(value #10) hair,
(address $41) food=(value #3) meat,
(address $58) color=(value #55) tawny,
(address $35) stripes=(value #8) black.

2. Store Facts in working Memory

The address represents the location in Working Memory. For example, address location 32 stores the value of 10.

3. Load Instruction

An instruction is loaded in ALU from Rule Memory. The first instruction of RULE 1 is a condition, and takes the form of:
(address $32) covering EQUAL (value #10) hair

4. Load Operands

a. Condition
The value of address location 32 is loaded into ALU, in this case 10. The comparison operation of ALU is:
(value #10) hair EQUAL (value #10) hair
This result is true
If the instruction is:
(address $32) covering EQUAL (value #11) feathers,
the output of ALU will be false. The control returns to STEP 3.
b. Action
If the instruction is an action such as:
(address $77) class MOV (value #20) mammal

ALU will get the value 20 and will store it at the address location 77.

5. Store Facts in Working Memory

The value of 20 is deduced from RULE 1 and is instructed to be stored at address location 77. The control returns to STEP 3.

6. Facts to Data

In this example the value at the (address $88) class is transferred to I/O interface. From the facts, the value at address location 88 is (value #100) tiger.

7. Data Output

The value 100 is translated by the interface to tiger.

Rule Base Structure

The application rule set 15 which is stored in working memory 16 is divided into two parts—STRUCT and RULES. A set of conditions in each rule is grouped together in adjacent addresses. Also, a set of actions in each rule is grouped together in adjacent addresses. These groups can be stored in the RULES part of working memory in the following fashion:

```
Rule #1
    address xxx1        condition_1_1
    address xxx2        condition_1_2
        .                   .
        .                   .
        .                   .
    address xxxm        condition_1_m
    address yyy1        action_1_1
    address yyy2        action_1_2
        .                   .
        .                   .
    1   .                   .
Rule #2
    address zzz1        condition_2_1
        .                   .
        .                   .
    .   .               .   .
```

Since conditions and actions are sequentially stored in different memory addresses, the representation of rules can be structured by using the starting address of each rule. Thus, the production rule can be expressed as:

```
if      xxx1
then    yyy1
if      zzz1
then    . . .
```

This format shows that if a group of conditions at a certain address is TRUE, then execute the group of actions at the address specified in then-part. Now, if the first rule fails then the control mechanism jumps to the starting address of the next rule. There is no need of the end-indicators for each rule, hence REX does not waste time on searching end-indicators.

Rule Base Structure of REX is illustrated in FIG. 4. For this version, External Memory of 64K×32 ROM is used to store the Application Rule Set 15. To maximize the utilization of limited memory, STRUCT and RULES are stored at both ends of Rule Memory 13, respectively. STRUCT starts from address OOOOH and increases. RULES starts from address FFFFH and decreases.

The detailed structure of Rule Memory is shown in FIG. 5. STRUCT stores the address index which points to the starting address of each rule in RULES. The size of Rule Memory is 64K, so only 16-bit lower-half word is used.

Each condition or action is represented as a 32-bit word instruction executed by REX. The condition is basically a logical comparison of two given operands. The actions are organized in a similar fashion. The operators of the actions are basically logic operators and an assignment operator. There are two operands for each operation: operand1 and operand2. Operand2 can be of two forms: direct or immediate. As shown in FIG. 4, the direct operand is a pointer to an address in the working memory represented by the symbol '$' and the immediate operand is an integer represented by '#'.

Instruction Set for REX Inference Engine

As shown in FIG. 5(b), instructions of REX are always 32-bit long. The Operation Code (6 bits), OP1 13 bits), and OP2 (13 bits) are assembled into one 32-bit instruction. Each rule in a given Application Rule Set has condition and action parts. Therefore, REX has two types of instruction set:

Condition Instructions: This type of instruction is used to check if the condition is True or False. This allows users to specify different logic relations between two operands, such as "Equal", "Greater Than", etc. The execution result of an Condition Instruction can only be True or False, which will affect the next execution sequence.

Action Instructions: This type of instruction is executed only when all the conditions of the current rule are True. The result of the execution of the action is always stored in the first operand.

The instruction and the corresponding operation codes are summarized in Table 1.

TABLE 1

REX OPERATION CODES

Operation Codes	Operation	Description
0X0000	EQ	Equal To; Is operand1 = operand2 ?
0X0001	NE	Not Equal to; Is operand1 <> operand2 ?
0X0010	GT	Greater Than; Is operand1 > operand2 ?
0X0011	LT	Less Than; Is operand1 <= operand2 ?
0X0100	GE	Greater than or Equal to; Is operand1 >= operand 2 ?
0X0101	LE	Less than or Equal to; Is operand1 <= operand2?
1X0000	NOT	logic NOT operand1; Each bit of the operand1 is complemented and the result is stored in operand1 in Working Memory
1X0001	AND	logic AND operand1 and operand2; Logic AND operation is performed on the correspondent bits of the operand1 and operand2. The result is stored in operand1 in Working Memory.
1X0010	OR	logic OR operand1 and operand2; Logic OR operation is performed on the correspondent bits of operand1 and operand2. The result is stored in operand1 in Working Memory.
1X0011	MOV	MOVe operand2 to operand1; The content of the operand2 is stored in operand1 in Working Memory.

TABLE 1-continued

REX OPERATION CODES

Operation Codes	Operation	Description
1X0100	SHR	SHift operand1 Right 1 bit; The least significant bit is discard and a zero is shifted into the most significant bit; the result is stored in operand1 in Working Memory.
1X0101	SHL	SHift operand1 Left 1 bits; The most significant bit is discard and a zero is shifted into the least significant bit; the result is stored in operand1 in Working Memory.
XX0110	JMP	JuMP to new address of External Memory; For JMP instruction, the least significant 16 bits of the instruction is loaded to C1 register which points to the new rule in External Memory.
XX0111	EOR	End of External Memory.

operand1 is direct-addressed data (WM[OP1]) from Working Memory.
operand2 can be direct-addressed data (WM[OP2]) or an immediate data (OP2).

The format of the opcode is displayed in FIG. 6. MSB (most Significant Bit), i.e. F1, of the opcode is used to specify the type of the instruction. If F1 is 0, it is a Condition instruction; otherwise it is an Action instruction.

A Condition instruction always has two operands. Whereas, an Action instruction may have only one or two operands depending on the operation needs.

REX allows two types of addressing mode: immediate and direct addressing. First operand always uses direct addressing mode. The second operand can be an immediate data or direct-addressed data. The addressing mode is distinguished by checking second MSB, i.e. F2, of the operation code. When F2 is 0, second operand is an immediate data. Otherwise, the second operand is a direct-addressed data.

Functional Description of the REX Chip

FIG. 7 provides a detailed block diagram of the REX chip 12. To avoid repetitive description, elements which have been previously described in connection with earlier drawing figures will be identified with the same reference characters.

Table 2 below lists the name, I/O type, and function of each input and output illustrated in FIG. 7.

TABLE 2

PIN DESCRIPTION OF REX

Symbol	Type	Name and Function
CLK	I	Clock Input: CLK controls the internal operations of REX chip. The maximum clock rate is 8 MHz.
$\overline{CS}$	I	Chip Select: Chip Select is an active low input used to select REX chip as an I/O device when CPU wants to read/write REX chip's internal registers (WM, WMC, C/S).
$\overline{EMCS}$	O	External Memory Chip Select: $\overline{EMCS}$ is low active. When REX is in inferencing mode, $\overline{EMCS}$ is used to select External Memory for information of the rule.

TABLE 2-continued

PIN DESCRIPTION OF REX

Symbol	Type	Name and Function
$\overline{IOR}$	I	I/O Read: $\overline{IOR}$ is low active. When both $\overline{CS}$ and $\overline{IOR}$ are active, CPU has read access to REX chip's internal registers (WM, WMC, C/S).
$\overline{IOW}$	I	I/O Write: $\overline{IOW}$ is low active. When both $\overline{CS}$ and $\overline{IOW}$ are active, CPU has write access to REX's internal registers (WM, WMC, C/S).
$\overline{READY}$	O	Ready: $\overline{READY}$ is low active. $\overline{READY}$ is a synchronization signal for external data transfer. $\overline{READY}$ goes low when REX is ready for new data.
RESET	I	Reset: RESET is high active. RESET is used to initialize REX chip state. All registers are reset after RESET is activated.
INT	O	INTerrupt Request: INT is high active. REX chip uses INT to interrupt CPU when REX chip finished the inferencing process.
A0-A1	I	Address: The two least significant address lines are used by CPU to control the data transfer to REX chip's internal registers (WM, WMC, C/S).
D0-D15	I/O	Data Bus: Data Bus lines are bidirectional three-state signals connected to system data bus. The Data Bus are output signals when $\overline{IOR}$ is active. The Data Bus are input signals when $\overline{IOW}$ is active.
MA0-MA15	O	External Memory Address Bus: When REX chip is in inferencing mode, External Memory Address Bus is used to address a rule in External Memory.
MD0-MD31	I	External Memory Data Bus: When REX chip is in inferencing mode, External Memory Data Bus sent the information regarding each rule to the REX chip.

WM: Working Memory
WMC: Working Memory Counter register
C/S: Control/Status flag registers

The identification of each register, and the function of each is as follows:

WM (Working Memory): Working Memory 16 is used to store the intermediate data during the inferencing process. Before REX starts the inferencing process, Working Memory is loaded with facts from user's input. The size of Working Memory limits the amount of user inputs to REX at any one time. In the illustrated embodiment, working Memory is a 4K×8 Static RAM.

WMC (Working Memory Counter) Register: WMC is an 13-bit increment counter with the capability of parallel load. During the I/O mode, WMC is used as Working Memory address counter for data transfer. When data transfer is proceeding, WMC will increment automatically. The content of WMC can be set by CPU before data transfer starts.

C1 Register: C1 is an 16-bit increment counter with the capability of parallel load. During the inferencing process, C1 points to one of the rules addresses in the STRUCT part of the Rule Memory 13. C1 increments by one before REX goes to the next rule. For JMP instruction, C1 will be loaded with a new value instead of incrementing by one.

C2 Register: C2 is an 16-bit decrement counter with the capability of parallel load. C2 points to the RULES part of Rule Memory. If no false condition occurs in a rule, C2 decrements by one before REX goes to the

next condition or action. When a false condition of a rule is detected, C2 will be loaded with the starting address of the next rule instead of decrementing by one.

OP Register: OP Register contains three parts: Operation Code, OP1, and OP2, which comprise an REX instruction. Operation Code is a 6-bit register that stores the operator of an instruction. Both OP1 and OP2 are 13-bit data registers that store the address of operand1 and operand2 in Working Memory respectively.

OP' Register: OP' Register is a prefetch Register used to store the prefetch instruction for OP Register. REX will execute the prefetch instruction except that when an JMP Instruction or a false condition occur.

SI (Start/Idle) Control Flag: SI is used to identify REX operation status: Inferencing Mode and I/O Mode. SI is set by CPU after the system sent all the facts to Working Memory. SI has the value 1 during the Inferencing Mode. SI is reset by REX each time the inferencing process stops, then REX switches to I/O Mode.

IE (Interrupt Enable) Control Flag: IE is set by CPU at the same time with SI flag. REX is granted the interrupt enable before REX goes to inferencing mode. IE is used with IRQ flag to generate interrupt signal. IE flag is reset by CPU at the end of the interrupt service routine.

IRQ (interrupt ReQuest) Status Flag: When inferencing process stops, IRQ is set by REX to indicate that REX is requesting an interrupt to CPU. IRQ is and-gated with IE flags to generate interrupt signal INT. IRQ is reset by CPU after the interrupt is acknowledged.

When REX is in I/O Mode, CPU can read or write REX registers. The signals and affected registers are listed in Table 3.

TABLE 3

DEFINITION OF REGISTER CODES

Register Operation	$\overline{CS}$	$\overline{IOW}$	$\overline{IOR}$	A1	A0
Read Status Registers	0	1	0	0	0
Write Control Registers	0	0	1	0	0
Read Working Memory Counter	0	1	0	0	1
Write Working Memory Counter	0	0	1	0	1
Read Working Memory	0	1	0	1	0
Write Working Memory	0	0	1	1	0
REX Chip is Not Selected	1	X	X	X	X

Operational Modes

REX has two operation modes:

I/O Mode

Inferencing Mode

Control flag SI is used as a mode flag. REX switches to the other mode when SI flag is changed.

Before REX get into Inferencing Mode, REX has to load all the user-input facts from the host system into Working Memory of REX. REX is switched from I/O Mode to Inferencing Mode when SI flag is set by host. After the inferencing process is terminated, the results will be transferred from Working Memory to the host system.

During the I/O operation, the host system can read or write specific registers when REX chip is selected. The control of read/write operations and the selection of registers are controlled by a set of control lines which are listed in Table 3. During reading and writing of WMC and C/S registers, only some bits of the system data bus are used. This is illustrated in FIG. 8.

Once Working Memory is loaded with user-input facts, REX will start the inferencing process from the first rule in External Memory. The inferencing flow of REX is shown in FIG. 9.

There are 3 different machine cycles for REX in inferencing Mode.

T1 Cycle: T1 is Rule Fetch Cycle. T1 cycle is executed only at the very beginning of the inferencing process or when JMP instruction occurs. T1 cycle fetches the starting address of a rule in External Memory to C1 register. C1 is actually a Rule counter, which points the starting address of currently inferenced rule.

Cycle: T2 is Instruction Fetch Cycle T2 cycle fetches the first Condition Instruction of each rule to REX registers. T2 cycle is executed when one of the conditions of a rule is false and the execution starts from the first instruction of the next rule. C2 can be regarded as an Instruction counter points to a Condition Instruction or an Action Instruction which is currently executed in ALU.

T3 Cycle: T3 cycle is Instruction Execution Cycle. There are several cases of the T3 cycle:

Condition Instruction/Immediate Data

Condition Instruction/Direct Addressing

Action Instruction/Immediate Data

Action Instruction/Direct Addressing

JMP

STOP (End of Rule)

The instruction prefetch cycle is overlapped with T3 cycle. If a JMP instruction occurs, execution sequence will go to T1 cycle. If the result of a Condition Instruction is false, the execution sequence will go to T2 cycle. If no JMP instruction and no false condition occurs, REX will use the prefetch data then go to T3 cycle.

REX will go through the same process over and over again, until all the rules in External Memory are inferenced. When inferencing process stopped, SI flag is reset to "0". Then REX switches from Inferencing Mode to I/O Mode.

Timing Chart

The timing charts for REX in the I/O Read Mode, the I/O Write Mode, and for external rule memory are shown in FIGS. 10-12 respectively. The A.C. (Alternating Current) characteristics of REX in I/O Mode is listed in Table 4.

TABLE 4

A.C. SPECIFICATION

Symbol	Parameter	Min	Typ	Max	Unit
TAS	I/O Address Setup Time	20	—	—	ns
TAH	I/O Address Hold Time	10	—	—	ns
TTW	I/O Read/Write Signal Width	60	—	—	ns
TOD	Data Output Delay Time	—	—	40	ns
TOH	Data Output Hold Time	10	—	—	ns
TDS	Data Setup Time	20	—	—	ns
TDH	Data Hold Time	10	—	—	ns
TRS	READY Signal Setup	0	—	—	ns

TABLE 4-continued

Symbol	Parameter	A.C. SPECIFICATION Min	Typ	Max	Unit
	Time				
TRD	READY Signal Delay Time	0	—	CLK*1	ns
TRW	READY Signal Width	CLK-10	CLK*1	CLK+10	ns
TMAW	External Memory Address Signal Width	CLK*2−20	CLK*2	CLK*2+20	ns
TMAC	External Memory Address Access Time	—	—	170	ns
TMOH	External Memory Data Output Hold Time	0	—	—	ns
TCSS	External Memory Chip Select Setup Time	0	—	—	ns
TCSH	External Memory Chip Select Hold Time	0	—	—	ns
TMOZ	External Memory Output Floating	—	20	—	ns

CLK is one cycle time of REX internal clock.

GLOSSARY

Antecedent: The *if* part of a production rule.

Application Domain: The subject or field to which the expert system pertains.

Application Rule: Set A set of rules, which are extracted by a knowledge engineer, pertaining to a specific application domain.

ASIC: Application Specific Integrated Circuit is a custom-designed integrated circuit for a specific application.

Consequent: The *then* part of a production rule.

CPU: Central Processing Unit: An operational unit which processes instructions and data.

Co-processor: A specialized processor which cooperates with a host computer to enhance the performance of the entire system.

Control Logic: A custom circuit that controls all the operations necessary for the REX chip.

DMA: Direct Memory Access: A commonly used communication method between a host computer and computer peripheral devices. DMA provides the most efficient way to transfer a block of data.

External Data: A block of binary data resides in a host computer memory.

External Memory: A physical memory which stores Application Rule Set.

Fact: A truth known by actual experience or observation. A group of facts are collected to combat conjectures.

Inferencing: Interpreting a rule of Application Rule Set.

Inference Engine: A problem-solving control mechanism for an expert system.

I/O Interface: A kind of device driver responsible for the communication between the computer host system and computer peripheral devices.

Knowledge Engineer: A person who extracts knowledge and facts of a particular application domain and converts them into Application Rule Set.

PC: Personal Computer.

PC/DOS: The Disk Operating System of Personal Computer, which managers the read/Write operations of a disk driver.

Production Rule: A rule specified in an *if-then* format.

RAM: Random-Access Memory: An electronic memory stores binary information which can be read-or-write-accessed.

ROM: Read-Only Memory: An electronic memory storage which stores the binary information. A ROM is read-accessed only; it does not have a write capability.

Rule Base Structure: An organization which stores the production rules in an efficient way to save the memory space and processing time.

Working Memory: A RAM that resides in the store the initial, intermediate, and final data of an inferencing process.

User Interface: A software program responsible for the communication between the end-users and the computer host system.

APPENDIX A
Example of Animal Identification Rule Set

```
RULE 1
IF
        (covering = hair)
THEN
        (class = mammal).
RULE 2
IF
        (produce = milk)
THEN
        (class = mammal).
RULE 3
IF
        (covering = feathers)
THEN
        (class = bird).
RULE 4
        (movement = fly) and
        (produce = eggs)
THEN
        (class = bird).
RULE 5
IF
        (food = meat)
THEN
        (carnivore = yes).
RULE 6
IF
        (teeth = pointed) and
        (limb) = claws) and
        (eyes = forward)
THEN
        (carvivore = yes).
RULE 7
IF
        (class = mammal) and
        (limbs = hoofs)
THEN
        (type = ungulate).
RULE 8
IF
        (class = mammal) and
```

-continued

APPENDIX A
Example of Animal Identification Rule Set

(food = cud)
THEN
(type = ungulate) and
(toed = even).
RULE 9
IF
(class = mammal) and
(type = carnivore) and
(color = tawny) and
(spots = dark)
THEN
(animal = cheetah).
RULE 10
IF
(class = mammal) and
(type = carnivore) and
(color = tawny) and
(stripes = black)
THEN
(animal = tiger).
RULE 11
IF
(type = ungulate) and
(neck = long) and
(legs = long) and
(spots = dark)
THEN
(animal = giraffe).
RULE 12
IF
(type = ungualte) and
(stripes = zebra).
THEN
(animal = zebra).
RULE 13
IF
(class = bird) and
(movement <> fly) and
(neck = long) and
(legs = long) and
(color = black_and_white)
THEN
(animal = ostrich).
RULE 14
IF
(class = bird) and
(movement <> fly) and
(swims = yes) and
(color = black_and_white)
THEN
(animal = penquin).
RULE 15
IF
(class = bird) and
(movement = flies_well)
THEN
(animal = albatross).

That we which claim is:

1. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first communications bus for communicatively connecting said working memory means to said logic means;

a second communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed, and for storing the deduced new facts in said memory means;

host computer means communicatively connected to said working memory means, for providing the facts pertaining to the application domain to said working memory means and for accepting the deduced new facts from said working memory means; and

wherein each of the instructions of said rule set includes an operator, a condition/action flag, and a pair of operands; and wherein said logic means includes an instruction decoder for testing said condition/action flag to determine whether the instruction is a condition or an action; means operable if the instruction is a condition for comparing the operands in accordance with the logical operation specified by the operator to generate a logic result; and means operable if the instruction is an action for performing the action specified by the operator on the operands.

2. The system of claim 1 wherein said logic means includes means operable if the logic result of said comparing means is TRUE for effecting fetching of the next instruction of the same rule.

3. The system of claim 1 wherein said logic means includes means operable if the logic result of said comparing means is FALSE for effecting fetching of the first instruction of the next rule.

4. The system of claim 1 wherein each of said instructions also includes a direct/immediate flag for specifying the addressing mode of one of the operands.

5. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first communications bus for communicatively connecting said working memory means to said logic means;

a second communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed, and for storing the deduced new facts in said memory means;

host computer means communicatively connected to said working memory means, for providing the facts pertaining to the application domain to said working memory means and for accepting the de-

duced new facts from said working memory means; and

wherein at least one of said instructions of each rule represents a condition to be satisfied by the facts of a given problem and including:

(i) an operation code defining a logical operation to be performed;

(ii) a first operand defining a first value to be compared by said logical operation; and

(iii) a second operand defining the address in said working memory containing a second value to be compared by said logical operation.

6. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first communications bus for communicatively connecting said working memory means to said logic means;

a second communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed, and for storing the deduced new facts in said memory means;

host computer means communicatively connected to said working memory means, for providing the facts pertaining to the application domain to said working memory means and for accepting the deduced new facts from said working memory means; and

wherein said rule set comprises a series of instructions in successive memory addresses beginning at one end of said rule memory means, and additionally including means for storing a rule index in successive memory addresses beginning at the opposite end of said rule memory means, the rule index comprising a series of memory addresses defining the beginning memory address of each rule of said rule set.

7. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first communications bus for communicatively connecting said working memory means to said logic means;

a second communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed, and for storing the deduced new facts in said memory means;

host computer means communicatively connected to said working memory means, for providing the facts pertaining to the application domain to said working memory means and for accepting the deduced new facts from said working memory means; and

wherein at least one of said instructions of each rule represents an action to be performed if all of the conditions of the rule are satisfied, and including:

(i) an operation code defining the action to be performed; and

(ii) a first operand defining a value for a fact, and

(iii) a second operand defining the address in said working memory means where the value defined in the first operand is to be stored.

8. The system of claim 7 wherein said means for successively executing instructions in said rule memory means comprises rule memory counter means including an address register for storing the address of the current instruction in said rule memory means; and means for updating said address register with the address of the next instruction each time an instructions fetched from said rule memory means.

9. The system of claim 7 further comprising means for loading the rule set into said rule memory means in forward chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform forward chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

10. The system of claim 7 further comprising means for loading the rule set into said rule memory means in backward chaining order; and wherein said logic means successively executes the instruction in said rule memory means to perform backward chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

11. The system of claim 7 further comprising means for loading the rule set into said rule memory means in combination chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform combination chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

12. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first communications bus for communicatively connecting said working memory means to said logic means;

a second communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed; and

wherein at least one of said instructions of each rule represents an action to be performed if all of the conditions of the rule are satisfied, and including:

(i) an operation code defining the action to be performed; and

(ii) a first operand defining a value for a fact, and

(iii) a second operand defining the address in said working memory where the value defined in the first operand is to be stored.

13. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first communications bus for communicatively connecting said working memory means to said logic means;

a second communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed; and

wherein said rule set is stored as a series of instructions in successive memory addresses beginning at one end of said rule memory means, and additionally includes means for storing a rule index in successive memory addresses beginning at the opposite end of said rule memory means, the rule index comprising a series of memory addresses defining the beginning memory address of each rule of said rule set.

14. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first bidirectional communications bus for communicatively connecting said working memory means and said logic means;

a second unidirectional communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed, for storing the deduced new facts in said working memory means; and

an input-output interface means, and a bidirectional input-output interface bus communicatively connecting said working memory means and said input-output interface means, for storing in said working memory from an external system, the facts pertaining to the application domain and for transferring the deduced new facts stored in said working memory to an external system; and

wherein at least one of said instructions of each rule represents a condition to be satisfied by the facts of a given problem and including:

(i) an operation code defining a logical operation be performed;

(ii) a first operand defining a first value to be compared by said logical operation; and

(iii) a second operand defining the address in said working memory a second value to be compared by said logical operation.

15. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first communications bus for communicatively connecting said working memory means to said logic means;

a second communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed; and

wherein at least one of said instructions of each rule represent a condition to be satisfied by the facts of a given problem and including:

(i) an operation code defining a logical operation to be performed;

(ii) a first operand defining a first value to be compared by said logical operation; and

(iii) a second operand defining the address in said working memory containing a second value to be compared by said logical operation.

16. The system of claim 15 wherein said means for successively executing instructions in said rule memory means comprises rule memory counter means including an address register for storing the address of the current instruction in said rule memory means; and means for updating said address register with the address of the next instruction each time an instruction is fetched form said rule memory means.

17. The system of claim 15 further comprising means for loading the rule set into said rule memory means in forward chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform forward chaining inferencing

upon the loaded rule set, and thereby deduce the new facts at high speed.

18. The system of claim 15 further comprising means for loading the rule set into said rule memory means in backward chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform backward chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

19. The system of claim 15 further comprising means for loading the rule set into said rule memory means in combination chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform combination chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

20. The system of claim 15 further comprising:

host computer means communicatively connected to said hardware-implemented rule-based expert system, for providing the facts pertaining to the application domain to said working memory means and for accepting the deduced new facts from said hardware-implement rule-based expert system.

21. The system of claim 15 further comprising:

host computer means communicatively connected to said memory means, for providing the facts pertaining to the application domain to said working memory means and for accepting the deduced new facts from said working memory means.

22. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first bidirectional communications bus for communicatively connecting said working memory means and said logic means;

a second unidirectional communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed, for storing the deduced new facts in said working memory means;

an input-output interface means, and a bidirectional input-output interface bus communicatively connecting said working memory means and said input-output interface means, for storing in said working memory from an external system, the facts pertaining to the application domain and for transferring the deduced new facts stored in said working memory to an external system; and

wherein at least one of said instructions of each rule represents an action to be performed if all of the conditions of the rule are satisfied, and including:

(i) an operation code defining the action to be performed; and

(ii) a first operand defining a value for a fact, and

(iii) a second operand defining an address in said working memory means where the value defined in the first operand is to be stored.

23. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first bidirectional communications bus for communicatively connecting said working memory means and said logic means;

a second unidirectional communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed, for storing the deduced new facts in said working memory means; and

an input-output interface means, and a bidirectional input-output interface bus communicatively connecting said working memory means and said input-output interface means, for storing in said working memory from an external system, the facts pertaining to the application domain and for transferring the deduced new facts storing in said working memory to an external system; and

wherein said rule set is stored as a series of instructions in successive memory addresses beginning at one end of said rule memory means, and additionally includes means for storing a rule index in successive memory addresses beginning at the opposite end of said rule memory means, the rule index comprising a series of memory addresses defining the beginning memory address of each rule of said rule set.

24. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action;

logic means;

a first bidirectional communications bus for communicatively connecting said working memory means and said logic means;

a second unidirectional communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for successively executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed, for storing the deduced new facts in said working memory means;

an input-output interface means, and a bidirectional input-output interface bus communicatively connecting said working memory means and said input-output interface means, for storing in said working memory from an external system, the facts pertaining to the application domain and for transferring the deduced new facts storing in said working memory to an external system; and

wherein each of the instructions of said rule set includes an operator, a condition/action flag, and a pair of operands; and wherein said logic means includes an instruction decoder for testing said condition/action flag to determine whether the instruction is a condition or an action; means operable if the instruction is a condition for comparing the operands in accordance with the logical operation specified by the operator to generate a logic result; and means operable if the instruction is an action for performing the action specified by the operator on the operands.

25. The system of claim **24** wherein said logic means includes means operable if the logic result of said comparing means is TRUE for effecting fetching of the next instruction of the same rule.

26. The system of claim **24** wherein said logic means includes means operable if the logic result of said comparing means is FALSE for effecting fetching of the first instruction of the next rule.

27. The system of claim **24** wherein each of said instructions also includes a direct/immediate flag for specifying the addressing mode of one of the operands.

28. The system of claim **24** further comprising means for loading the rule set into said rule memory means in backward chaining order; and wherein said logic means successively executes the instruction in said rule memory means to perform backward chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

29. The system of claim **24** further comprising means for loading the rule set into said rule memory means in combination chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform combination chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

30. The system of claim **24** further comprising:

host computer means communicatively connected to said input-output interface means, for providing the facts pertaining to the application domain to said input-output interface means and for accepting the deduced new facts from said input-output interface means.

31. The system of claim **24** wherein said means for successively executing instructions in said rule memory means comprises rule memory counter means including an address register for storing the address of the current instruction in said rule memory means; and means for updating said address register with the address of the next instruction each time an instruction is fetched from said rule memory means.

32. The system of claim **24** further comprising means for loading the rule set into said rule memory means in forward chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform forward chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

33. A hardware-implemented rule-based expert system apparatus suitable for performing high speed inferencing based upon a rule set for an application domain, comprising:

logic means;

working memory means for storing therein facts pertaining to the application domain;

rule memory means for storing therein the rule set for the application domain, comprised of a series of instructions each defining a condition or an action each of the instructions of said rule set including an operator, a condition/action flag, and a pair of operands;

a first communications bus for communicatively connecting said working memory means to said logic means; and

a second communications bus for communicatively connecting said rule memory means to said logic means;

said logic means comprising means for success executing the instructions in said rule memory means obtained via said second communications bus, with reference to the stored facts in said working memory means obtained via said first communications bus, to thereby deduce new facts at high speed; said logic means further comprising an instruction decoder for testing said condition/action flag to determine whether the instruction is a condition or an action; means operable if the instruction is a condition for comparing the operands in accordance with the logical operation specified by the operator to generate a logic result; and means operable if the instruction is an action for performing the action specified by the operator on the operands.

34. The system of claim **33** further comprising:

host computer means communicatively connected to said working memory means, for providing the facts pertaining to the application domain to said working memory means and for accepting the deduced new facts from said working memory means.

35. The system of claim **33** further comprising:

host computer means communicatively connected to said hardware-implement rule-based expert system, for providing the facts pertaining to the application domain to said working memory means and for accepting the deduced new facts from aid hardware-implemented rule-based expert system.

36. The system of claim **33** wherein said means for successively executing instructions in said rule memory means comprises rule memory counter means including an address register for storing the address of the current instruction in said rule memory means; and means for updating said address register with the address of the next instruction each time an instruction is fetched from said rule memory means.

37. The system of claim **36** wherein said logic means further comprises means for storing the deduced new facts in said working memory means; wherein said first communications bus comprises a bidirectional communications bus for loading the facts pertaining to the application domain from said working memory means into said logic means and for storing the new facts deduced by said logic means into said working memory means; and wherein said second communications bus comprises a unidirectional communications bus for loading the instructions defining a condition or an action from said rule memory means into said logic means.

38. The system of claim 33 wherein said logic means further comprises means for storing the deduced new facts in said working memory means; and wherein said system further comprises output means for transferring the deduced new facts stored in said working memory to an output device.

39. The system of claim 38 wherein said output means comprises an input-output interface and an input-output interface bus communicatively interconnecting said input-output interface to said working memory means.

40. The system of claim 33 wherein said rule set is stored as a series of instructions in successive memory addresses beginning at one end of said rule memory means, and additionally includes means for storing a rule index in successive memory addresses beginning at the opposite end of said rule memory means, the rule index comprising a series of memory addresses defining the beginning memory address of each rule of said rule set.

41. The system of claim 33 wherein said logic means includes mean operable if the logic result of said comparing means is TRUE for effecting fetching of the next instruction of the same rule.

42. The system of claim 33 wherein said logic means includes means operable if the logic result of said comparing means is FALSE for effecting fetching the first instruction of the next rule.

43. The system of claim 33 wherein each of said instructions also includes a direct/immediate flag for specifying the addressing mode of one of the operands.

44. The system of claim 33 further comprising means for loading the rule set for the application domain into said rule memory means.

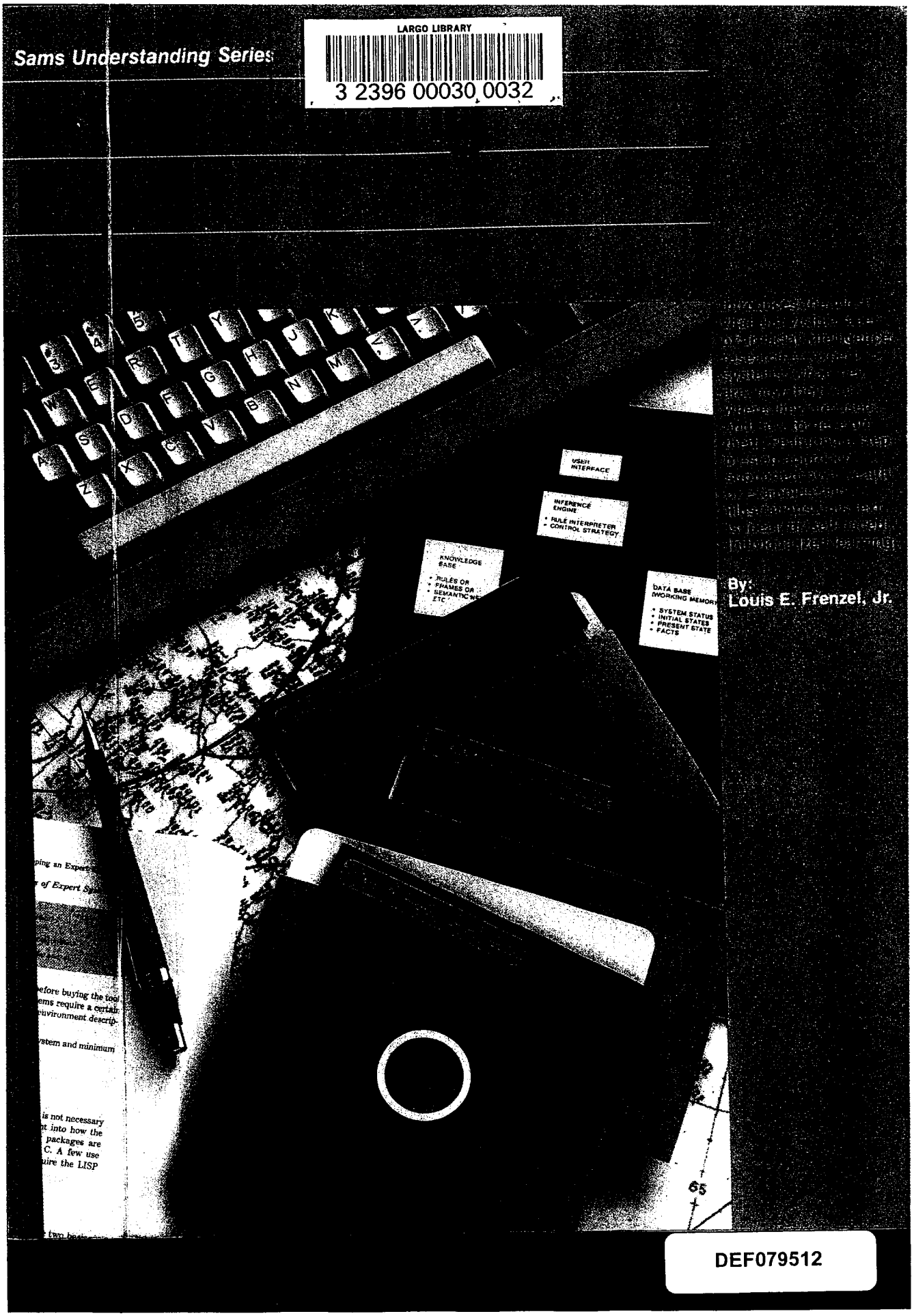
45. The system of claim 44 wherein said means for loading the rule set for the application domain into said rule memory means comprises means for loading the rule set into said rule memory means in combination chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform combination chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

46. The system of claim 44 wherein said means for loading the rule set for the application domain into said rule memory means comprises means for loading the rule set into said rule memory means in forward chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform forward chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

47. The system of claim 44 wherein said means for loading the rule set for the application domain into said rule memory means comprises means for loading the rule set into said rule memory means in backward chaining order; and wherein said logic means successively executes the instructions in said rule memory means to perform backward chaining inferencing upon the loaded rule set, and thereby deduce the new facts at high speed.

* * * * *

Sams Understanding Series



By:
Louis E. Frenzel, Jr.

FIRST EDITION
FIRST PRINTING—1987

International Standard Book Number: 0-672-27065-X
Library of Congress Catalog Card Number: 87-72206

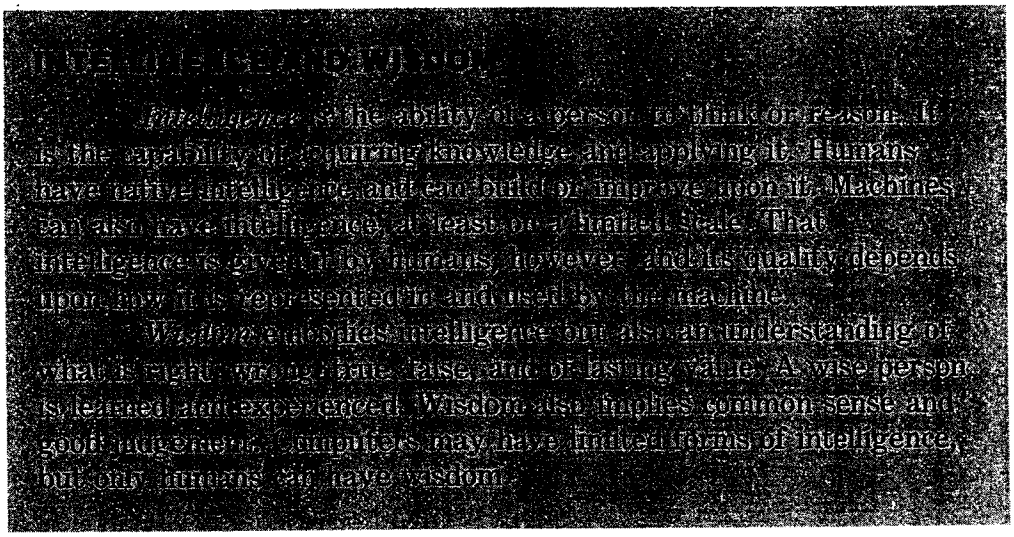
Acquisitions Editor: Greg Michael
Manuscript Editor: Don MacLaren
Cover Art: Diebold Glascock Advertising Inc.
Cover Photography: Castle Productions, Inc.
Components Courtesy of: WRTV, Indianapolis
Illustrator: Don Clemons
Indexer: Sandi Schroeder
Compositor: Shepard Poorman Communications Corp.

Printed in the United States of America

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks are listed below. In addition, terms suspected of being trademarks or service marks have been appropriately capitalized. Howard W. Sams & Co. cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Ada is a registered trademark of the U.S. Government Ada Joint Program Office.
DEC is a registered trademark of Digital Equipment Corporation.
EXSYS is a registered trademark of EXSYS, Inc.
1st Class is a trademark of Programs in Motion, Inc.
IBM is a registered trademark of International Business Machines Corporation.
VAX is a registered trademark of Digital Equipment Corporation.



Conventional vs. AI Computing

Expert systems and other forms of AI are software that can be made to work on almost any digital computer.

Artificial intelligence and expert systems are software. They are a collection of programs that make a computer think, reason, and apply the knowledge stored in them. Computers are general-purpose boxes of electronic components that are designed to process information in some way. By themselves, they are worthless. But given appropriate software, computers become those powerful tools that many have learned to love or hate.

Most computers use what we can call conventional software. These are the programs that solve math problems, manipulate data bases, compute spreadsheets, spell-check a document, among other functions. AI software is different from these. It uses a totally new approach to solving a problem, called "symbolic processing," in which the computer is given reasoning capability that simulates what a human might do. Let's examine the basic kinds of computing.

Conventional Computing

An algorithm describes ingredients and how to combine them, like a recipe.

Figure 1-1 lists most of the ways computers process data. Conventional software is used to perform all of these operations. To implement these functions, programs are written that tell the computer specifically what to do. The computer is given a step-by-step sequence called an "algorithm," which clearly defines the actions that must be taken to solve a given problem.

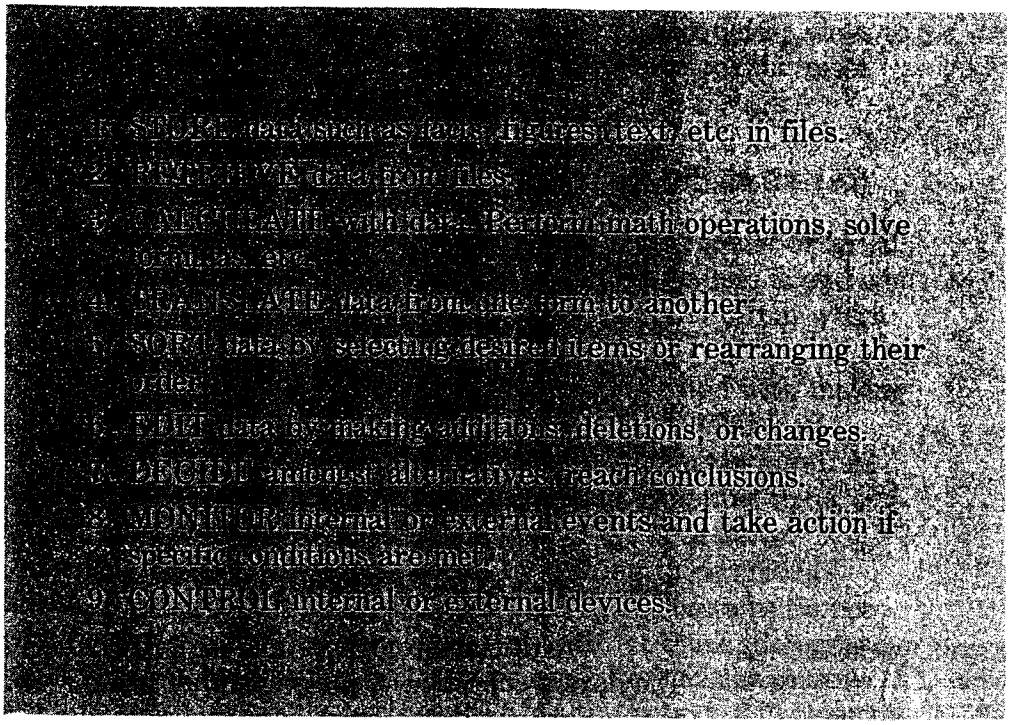


Figure 1-1.
How Conventional Software Programs Process Data on Computer

To solve a conventional computer problem, a programmer first analyzes the problem to determine exactly what the inputs are and what the desired outputs should be. The programmer then comes up with an algorithm that processes the inputs to produce the output. That algorithm is a clearly defined incremental procedure that does the desired manipulation. The programmer then takes this algorithm and converts it into a sequential list of instructions, statements, or commands as defined by a programming language. In turn, that programming language produces the binary code that is stored in the computer memory. When executed, the program solves the problem exactly as specified.

Figure 1-2 shows a block diagram of a general-purpose digital computer and how the computer performs conventional computing. The algorithm is stored in the computer memory along with the data to be processed. The algorithm manipulates the input data to produce the desired output.

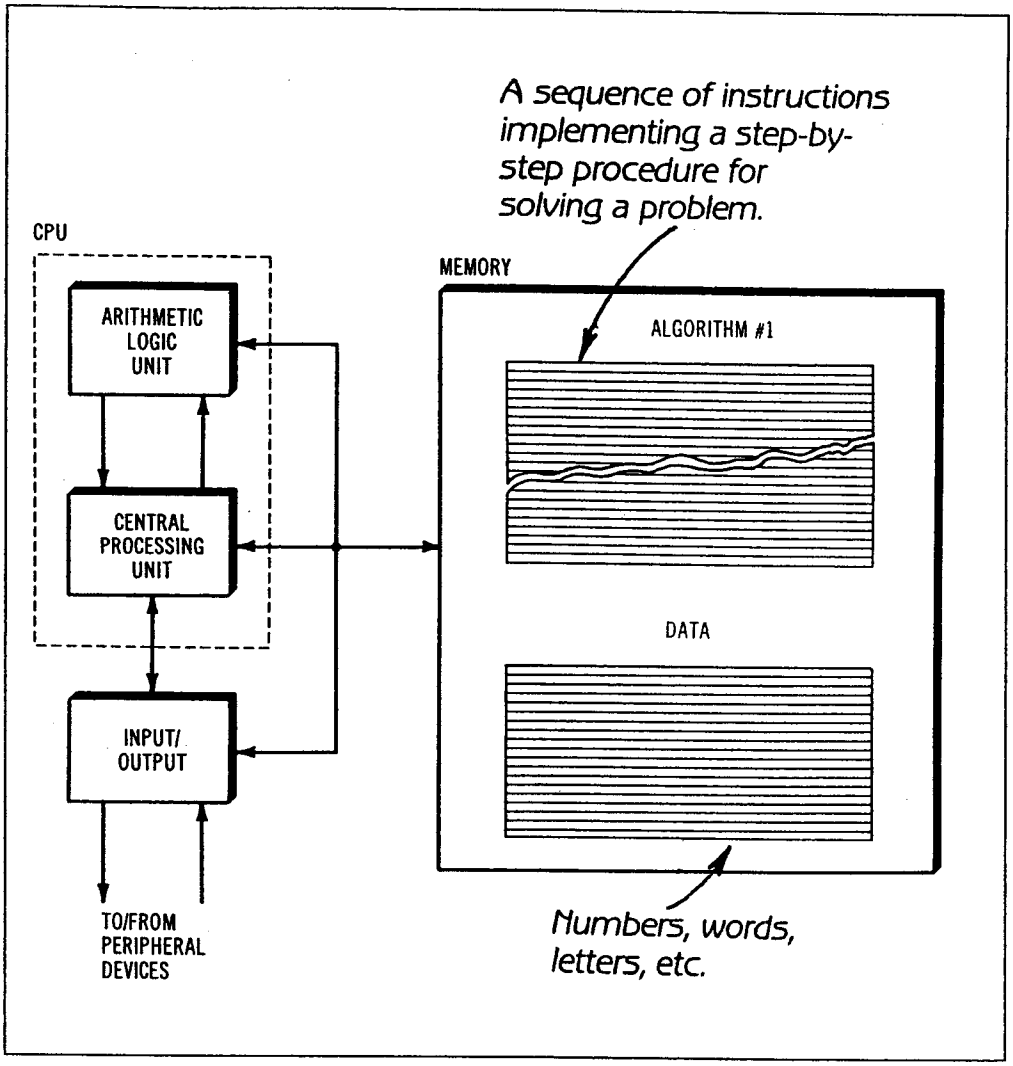


Figure 1-2. How Computer Processes Conventional Algorithmic Software

AI Computing

A symbol is something that represents something else.

AI, or symbolic, computing uses a totally different approach to problem solving. It starts with knowledge of the domain. This knowledge must be represented in a form that can be stored in a computer. The knowledge is represented symbolically. A symbol is nothing more than a word, letter, or number that you use to represent objects, actions, and their relationships. These objects can, of course, represent anything, including people, places, events, and ideas. These symbols can be stored in the computer memory as ASCII characters or strings. Through the use of symbols, you can create a *knowledge base* which states various facts about the objects, actions, or processes, and how all of them are interrelated.

The two main parts of any AI software are a knowledge base and an inferencing program.

Once the knowledge base has been created, a method must be devised to use it. Basically. a program is needed that will use the knowledge to reason and think in an effort to solve a particular problem. This kind of program is generally referred to as an "inferencing program," or "inferencing engine," which is designed to make decisions and judgements based upon the symbolic data in the knowledge base. The inferencing program accepts external inputs about the problem and then attempts to apply the available knowledge to its solution.

The inferencing program manipulates the symbolic information in the knowledge base through a process of search and pattern-matching. The inferencing program is provided with some initial inputs that provide sufficient information for the program to begin. Using these initial inputs, the inferencing program searches the knowledge base looking for matches. The search continues until a solution is found. The initial search may turn up a match that will, in turn, lead to another search and another match, and so on. The inferencing program performs a series of these searches that are chained together. This chain simulates a logical reasoning process.

The basic problem-solving approach of any AI program is search and pattern-matching.

All AI programs use this search and pattern-matching approach, looking for links and relationships. This process may solve the problem satisfactorily; in some cases, though, the problem may not be solved at all. If insufficient knowledge is available or insufficient input data is given, symbolic computing may not be able to solve the problem. This is in contrast to conventional computing where an algorithm always produces a solution if given input data to process. However, when enough input is provided and a good knowledge base exists, very satisfactory solutions usually result from symbolic computing. The effect is as if a human had solved the problem.

The inferencing program is implemented with algorithms that define the search and pattern-matching techniques to be used on the symbolic knowledge base. It is these techniques that solve the problem, not the algorithms. The simplified block diagram of a digital computer in *Figure 1-3* shows how the knowledge base and inferencing programs are stored in memory.

AI software tells "what" but not "how."

Conventional software does procedural processing in which the algorithm details specifically how to solve the problem. AI software, including expert systems, does non-procedural processing. The program contains the "what" of the problem but not the "how." The "what" is the knowledge and any input data. But no procedure manipulates this "what" according to a step-by-step "how." Instead, through its search and matching processes, the inference program finds its own conclusion.

Types of Expert Systems

ABOUT THIS CHAPTER

In this chapter, we'll outline the types of expert systems in use and we'll examine the kinds of problems that expert systems solve best. When you complete this chapter, you'll know how expert systems are being used and you'll have a feel for whether an expert system might be of value to you.

EXPERT SYSTEM CONFIGURATIONS

There are a variety of ways that expert systems can be set up and used. These configurations include stand-alone, hybrid or embedded, linked, dedicated, and real-time systems.

Stand-Alone Systems

Stand-alone expert systems are pure AI and run by themselves on the computer.

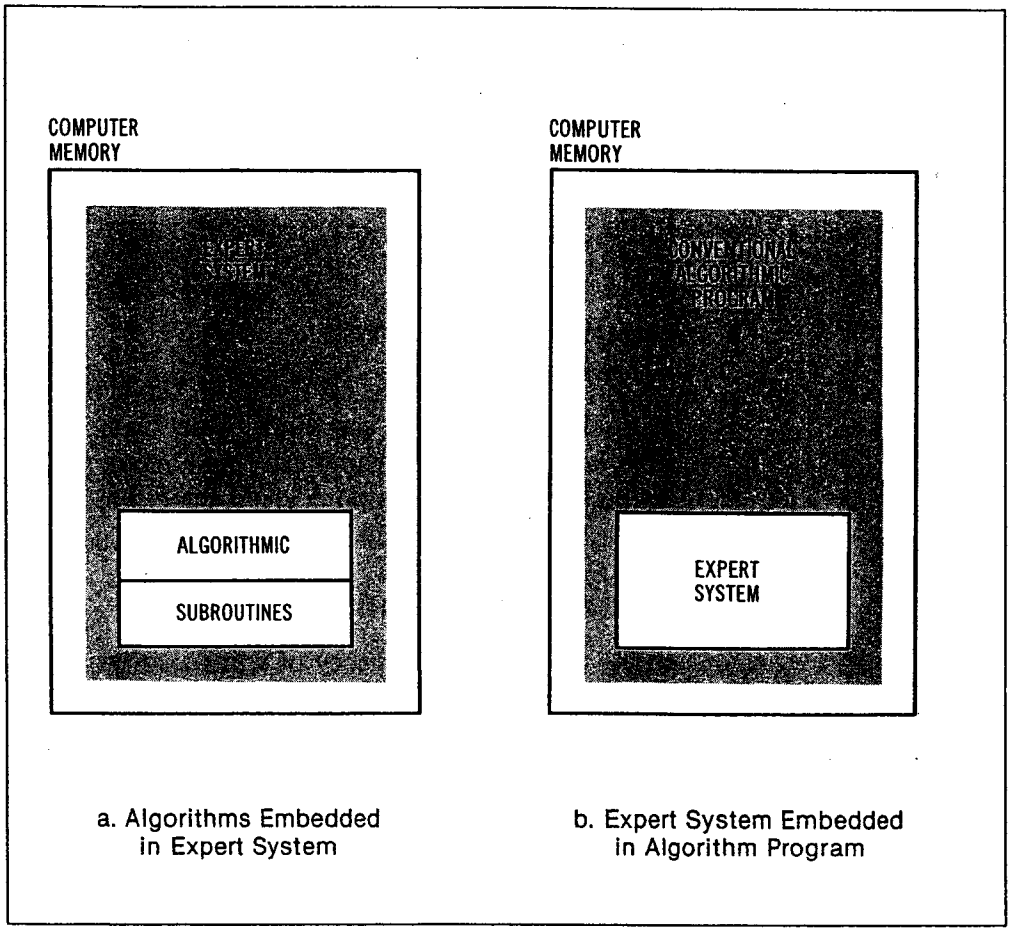
A stand-alone expert system is one that runs by itself and fully occupies its host computer. The program is loaded from a floppy disk or transferred from a hard disk into the computer memory and then executed. Most of the expert systems that you will encounter will be of this type.

Hybrid Programs

Conventional and AI programs may be mixed to form embedded or integrated software.

Expert systems that are embedded in or integrated with algorithmic routines are called hybrid programs. The two main types of embedded or integrated systems are illustrated in *Figure 2-1*. In *Figure 2-1a*, an expert system is the main program but it has embedded in it algorithmic subroutines. To perform its problem-solving function, the expert system may need to perform some calculations or do other jobs best assigned to algorithmic routines. The algorithms are buried in the expert system software, which refers to them when necessary.

Figure 2-1.
Two Kinds of Hybrid Programs



It is also possible to embed an expert system in a conventional algorithmic program, as *Figure 2-1b* shows. The main processing function is carried on by the conventional program, with occasional reference to the embedded expert system as the computing may require. For example, you could have an expert system that made critical accounting decisions based upon computations being made in a conventional general ledger or accounts payable program. The program would call the inferencing program of the expert system when such a decision was required.

Linked Software

Expert systems may be linked with one another or with conventional programs to get the inputs they need to solve problems.

Another form of mixed system is one that links conventional and AI programs or multiple expert systems. Commonly, the expert system requires input data from another source in order to solve the problem or make its decisions. Many expert systems are set up with links to external software packages such as spreadsheets or data base management systems. In this way, the expert system can

An expert system is just a tool. It's up to the user to make the final decision.

But you are still in control. The expert system may give you an answer, conclusion, or recommendation, but the final decision is up to you. If you don't believe or trust the output, ignore it and draw your own conclusions. Use your own judgement as to when to apply an expert system's output.

WHAT EXPERT SYSTEMS ARE NOT

Just because a program solves a problem or makes a decision doesn't mean that it is an expert system.

Whether a program is an expert system often isn't germane to the user. The user cannot tell whether heuristic symbolic processing is going on or whether an algorithm is being solved. For this reason, expert systems are sometimes confused with other types of software that appear to perform similar functions, including data base management systems, decision software, and decision support systems (DSS). All of these use conventional algorithmic approaches to problem-solving even though the kinds of problems solved may be similar to those solved by expert systems.

Data Base Management Systems (DBMS)

A DBMS is a collection of related facts stored in a computer for fast, convenient retrieval.

A data base management system is a software package that allows a user to create and manage large files of data, called a "data base." A data base is sometimes confused with a knowledge base. A data base comprises small units of data called records, which comprise individual data elements called fields. For example, in a customer record, the name, address, city, state, zip code, and telephone number may each represent a field. A record in an inventory file may have fields representing part name and number, source, quantity on hand, and cost. *Figure 2-4* shows the basic structure of a DBMS.

While a record is a unit of data containing facts and figures rather than knowledge, a knowledge base comprises individual "chunks" of knowledge. A common form of expressing knowledge is called a "rule." A typical rule might be "*If* the regulator output voltage is zero *and* the input to the regulator is normal *then* the regulator is defective."

You can readily see the difference between an element of data and an element of knowledge. We'll examine rules in Chapter 3.

or field. These are usually output for further analysis. The output of the expert system is an answer or conclusion drawn by applying the knowledge in the knowledge base to the problem.

While there are similarities between the two, the kinds of problems they solve are considerably different. DBMS and other types of file-management software are used primarily to store great volumes of information and access them readily. The data may be useful by itself, but it must be understood, analyzed, and interpreted by a human in order for it to be of value.

On the other hand, the expert system provides some kind of an output conclusion. Working with input data, it uses the knowledge base to understand the problem sufficiently to make a recommendation or decision. It may even provide an explanation subsystem to explain the decision.

Decision Software

Mathematical techniques can be used to help make tough decisions.

There is a special kind of algorithmic software designed to help individuals make decisions. These programs examine data given to them and, like expert systems, suggest an optimum decision or conclusion. Because they do not use symbolic representation or use search and pattern-matching as their basic method of manipulation, they aren't considered expert systems or AI programs. Such decision-making programs are implemented with a number of widely known mathematical decision algorithms and, in use, you cannot easily distinguish a decision-making program from an expert system.

Decision software can help you make a decision on a complex problem. Typically, the goal is to choose the best of several different alternatives. The criteria for selecting a particular outcome are defined in numerical terms. Their importance or priority is weighted, and by converting the criteria into numerical values, mathematical algorithms can be used to process them and select the most desirable outcome. Decision software provides a precise solution, whereas an expert system may offer only an approximate solution, or a guess, or no solution at all.

Decision Support Systems (DSS)

DSS helps managers run large, complex organizations by providing ways to analyze data, understand it, and make decisions.

A decision support system (DSS) is a collection of programs used for decision-making. Such programs help management in forecasting, planning, and managing large enterprises. Decision support systems usually incorporate some kind of decision software.

Many DSS also include a modeling capability that enables a mathematical simulation of a situation to be built in order to test various tactics and strategies. The model is usually a mathematical

relationships, and classify objects. Using declarative knowledge you cannot explain anything, but you can present truths and their association with each other. In expert systems, declarative knowledge representation schemes include semantic networks, frames, and production rules.

Procedural Knowledge

Procedural or *prescriptive* knowledge, is explanatory; it provides a way of applying the declarative knowledge. With this kind of knowledge, you can show procedures for performing a course of action. Procedural knowledge recommends what to do and how. A list of instructions for installing a program on a hard disk storage unit and a step-by-step sequence for disassembling a large electric motor are both examples of procedural knowledge. Procedural knowledge is represented in expert systems as production rules and scripts.

PRODUCTION RULES

The common method of representing knowledge in expert systems is with production rules. Production rules, sometimes referred to as "rules" or "productions," are two-part statements that contain a small increment of knowledge. The domain, or subject, to be represented in an expert system is divided up into many small chunks of knowledge. The knowledge is usually heuristic, which is easily represented in rule format. Expert systems using production rules are sometimes called "production systems," and the knowledge base is sometimes called the "rule base." All production systems are expert systems, but not all expert systems are production systems because there are forms of knowledge representation other than production rules.

Rule Format

A rule is a two-part statement containing a premise and a conclusion.

The two parts of a rule are a premise and a conclusion, a situation and an action, or an antecedent and a consequent. These statements are written in an *IF-THEN* format. The first part of the rule, generally called the "left-hand" part, is prefaced by the word *IF*, to state a situation or premise. The second or "right-hand" part of the rule is prefaced with *THEN* to state an action or a conclusion. Production rules are simple to understand and use and are ideally suited to a wide range of heuristic knowledge. Most knowledge domains are easily represented in this format. Some examples of rules are shown here.

1. IF the water level exceeds 4 feet
   THEN start the pump.
2. IF the fish has a triangular top fin
   THEN it is a shark.

Some types of knowledge require a more complex rule, such as this:

3. IF the fruit is red
   AND has seeds
   AND grows on a vine
   THEN it is a tomato.

In this rule, the premise or situation begins with an IF but also contains two AND statements that are part of the situation or premise. If the conclusion is to be true, then all three statements in the premise must be true. As many additional AND statements may be used as are required to represent a desired piece of knowledge.

Another way to make a knowledge statement is to use OR statements in the premise. Along with the initial statement, one or more OR statements may also be included.

4. IF the switch is off
   OR the fuse is blown
   THEN the circuit will be off.

In rules of this type, the conclusion stated in the THEN part of the rule will be true IF any one or more of the statements in the premise is true. This format provides a flexible way of representing some types of knowledge. More complex rules may include AND and OR statements.

Since rules represent only tiny increments of knowledge, it takes a considerable number of them to represent the knowledge of a particular domain. Small expert systems may have only ten or twenty rules but the more useful systems usually have well over a hundred. Large systems have many thousands of rules.

The main benefit of rules is that they facilitate creation, modification, and maintenance of a knowledge base because the knowledge is modularized. Since much domain knowledge changes over time, new rules must be added and old rules removed or modified to keep the knowledge base current. With rules, these changes can be made quickly and easily.

Measures of Confidence

Not all knowledge or decisions are exact. Expert systems can handle that.

We mentioned earlier that AI software is capable of dealing with ambiguity and uncertainty. Algorithmic software is not. An algorithm, such as a formula, needs specific input values supplied before it can compute an output. If you give it proper inputs, it will generate a correct output. Of course, if the inputs are wrong, you can expect to get an incorrect output and thus the old expression "Garbage in, garbage out."

AI software, however, doesn't always need perfect inputs and outputs in order to be useful. When the expert system you are using asks you questions, you may not be able to supply the desired answers. In some cases, you will not know the answers at all. There will be other times when you have answers but you're unsure of their validity. Most expert systems are designed to be able to deal with these situations.

Certainty Factors

By using certainty factors, expert systems can handle ambiguous and uncertain knowledge.

One method that has been devised to deal with uncertainty is *certainty factors*. A certainty factor (CF) is a numerical measure of the confidence you have in the validity of a fact or rule. It allows the inferencing program to work with inexact information.

A variety of certainty factor scales can be used. Some of these are illustrated in *Figure 4-1*. The most often used is a scale from 0 to 1 where 0 indicates a total lack of confidence and 1 represents complete confidence. Other expert systems may use such scales as 0 to 10 or 0 to 100. You can also use a −1 to +1 scale as *Figure 4-1* shows. Other arbitrary arrangements may be set up by a programmer. Here is a rule for using certainty factors:

IF the regulator output is zero
AND the regulator input is correct
THEN the regulator circuit is defective (.9).

This rule states that we are pretty sure that if the input is good but the output is zero, then the problem is in the regulator. But we don't give the conclusion a 1.0 confidence rating because there could conceivably be a less common problem, say a broken wire or defective connector.

While the programmer sets up the CF scale, the expert actually puts the correct value on the rule. Only the expert knows just how confident he or she is in the rule's outcome because certainty factors are nothing more than intelligent guesses based

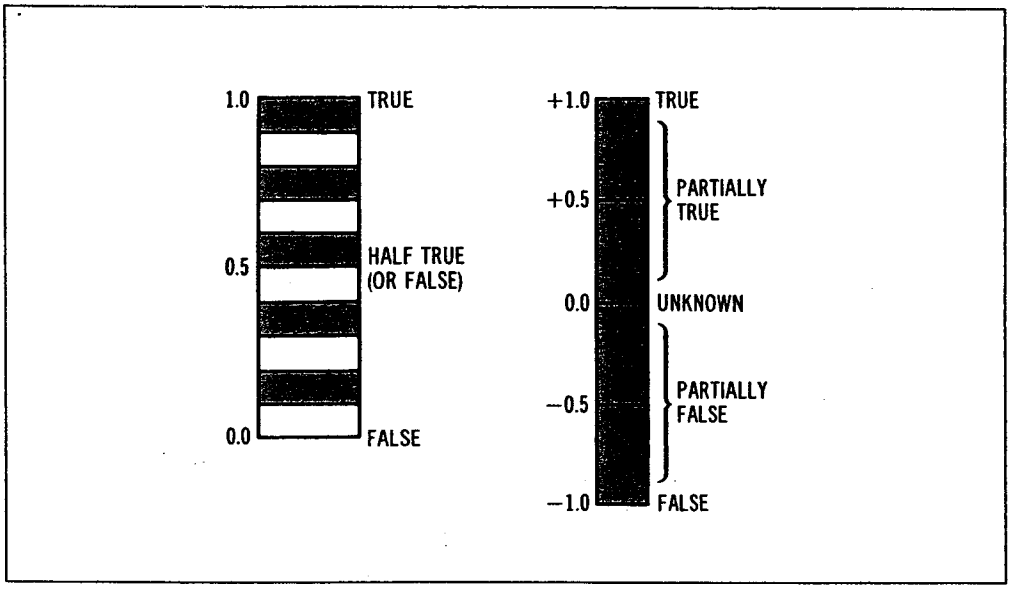


Figure 4-1.
Examples of Certainty Scales

upon experience and available statistical data. The expert sets the certainty factors when constructing the knowledge base, but they may have to be changed when the system's validity is tested.

The expert system usually reaches a conclusion based on several rules in a chain. If each rule or conclusion has a CF, the outcome will have a composite CF. Several methods have been devised for combining certainty factors. One of these is indicated here.

$$CF(C) = CF(A)+CF(B)-CF(A)\times CF(B)$$

The certainty factor of the conclusion or outcome (C) depends upon the certainty factors of rules A and B, as we see here:

Rule A: IF X
AND Y
THEN Z(CF = .75).

Rule B: IF D
AND E
THEN F(CF = .3).

$$\begin{aligned} CF(C) &= CF(A)+CF(B)-CF(A)\times CF(B) \\ &= .75+.3-(.75)(.3) \\ &= 1.05-.225 \\ &= .825 \end{aligned}$$

Probability

Probability is the ratio of the number of outcomes to the number of events causing them.

It is important to understand that a certainty factor is not the same thing as *probability*. A CF is just a number on an arbitrary scale that states to what extent we believe the knowledge is true. Probability, on the other hand, is a number that indicates the chance of an action occurring or not occurring. Probability (P) is the ratio of the number of times that an event (X) occurs to the total number of events (N) that take place. Mathematically, this is expressed as:

$$P = X/N$$

For example, when you roll a die, you know that there is one chance in six of the die landing on a particular number. If the die is perfectly balanced and it is thrown a great number of times, each of the six sides would eventually show up one-sixth of the time. The chance of rolling a two, for example, expressed mathematically is simply:

$$P(2) = 1/6 = .16666, \text{ or about } 16.7\% \text{ of the time}$$

Bayesian probability is used in some expert systems to aid in working with inexact data and decisions so common in the real world.

Depending upon the type of knowledge involved, probability may be a more suitable way to deal with uncertainty than certainty factors so some expert systems use probability rather than certainty factors. In general, these are harder to implement.

One popular kind of probability calculation is Bayes' theorem or rule. This Bayesian probability is a formula that computes the probability of event X occurring if event Y has already occurred. In probability math terminology, this is P(X:Y). It is computed with the expression:

$$P(X:Y) = \frac{P(Y:X)P(X)}{P(Y:X)P(X)} + P(Y:\text{not } X)P(\text{not } X)$$

P(not X) means the probability of X not occurring and is simply equal to 1−P(X) or 1 minus the probability that X does occur. P(Y:X) means the probability of Y occurring, given that X has already occurred.

Stringing such calculations along in a big system where a lot of rules are evaluated to reach a conclusion causes a lot of computing to take place, slowing down the process. Despite its complexity, such an approach works if warranted. The PROSPECTOR expert system for locating mineral deposits uses this approach.

How Expert Systems Work

ABOUT THIS CHAPTER

So far you've seen what expert systems are and how they can be used. We've explored ways that knowledge is represented in AI software and how search techniques can be applied in order to simulate human thinking. Now, let's put all of these things together and find out how expert systems work.

We know that most AI software consists of two main parts, a knowledge base and an inferencing program. Expert systems certainly have these, but some other elements are required to make them useful. These include a global data base, a user interface, an explanation facility, and a method of entering new knowledge. We'll take a look at each of these major components in this chapter.

Finally, we'll also take an in-depth look at how a rule-based expert system goes about making its decisions and arriving at a conclusion. We'll explore forward- and backward-chaining rule-based expert systems to get a feel for what actually goes on during a consultation.

EXPERT SYSTEM ARCHITECTURE

The main elements of all expert systems are a knowledge base, an inferencing capability, a data base, and a user interface.

Figure 5-1 shows a block diagram of an expert system. These are the main elements of an expert system, although not all expert systems have every subsection. All expert systems have the knowledge base, inference engine, data base, and user interface, but other features will vary from program to program.

Knowledge Base

The knowledge base contains the domain expertise. Most expert systems use production rules and, therefore, the knowledge base is often referred to as a *rule base.* Some expert systems also use semantic networks and frames.

A key feature of an expert system is that the knowledge base is independent of the inference engine that accesses it to solve problems. The knowledge base is formatted in accordance with the

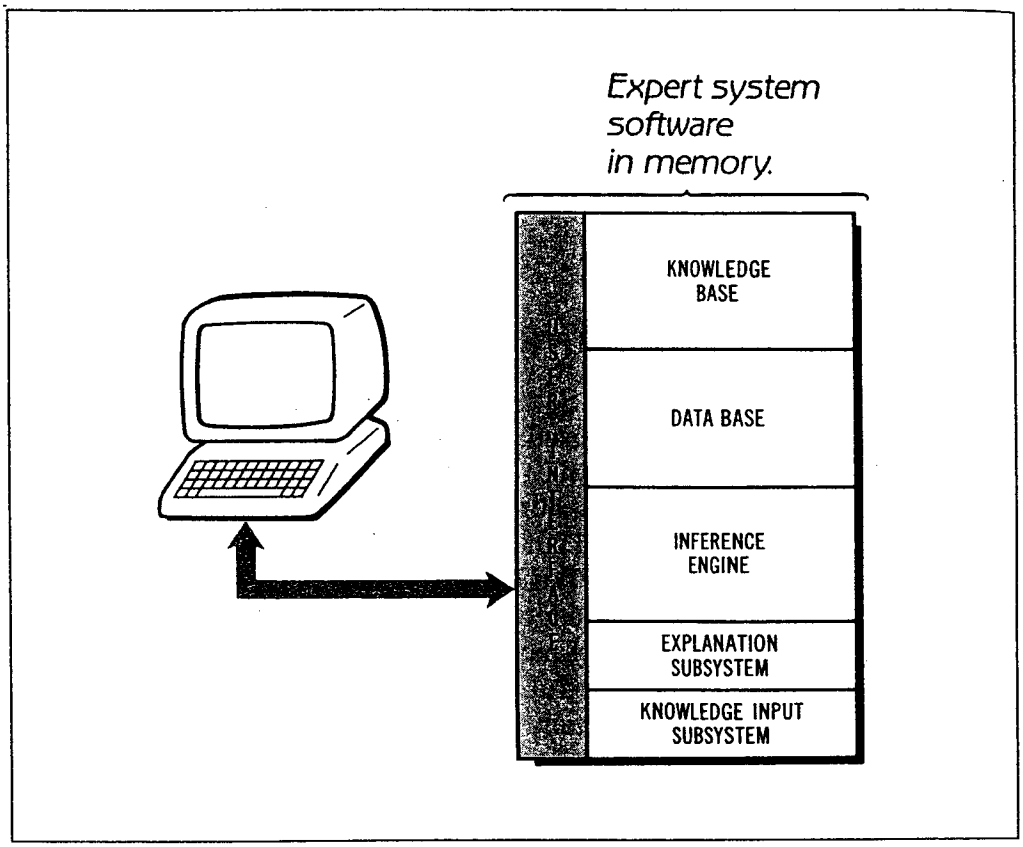


Figure 5-1.
General Block Diagram of an Expert System

nature of the knowledge. That format is known by the inference engine so that it can access the IF and THEN portions of the rules separately as the problem-solving strategy requires.

Since the knowledge base in rule-based systems is independent of the inferencing program, adding knowledge is fast and easy.

Since the knowledge base is separate from the algorithmic programs of the inference engine, should new information become available or knowledge stored become obsolete, it is relatively easy to make the necessary changes. All that you have to do is add new rules, remove old rules, or correct existing rules; reprogramming is not necessary.

There are no specific guidelines for storing rules in a knowledge base. In fact, because the inference engine performs a search of the rule base, the rules may be stored in virtually any order. Although the rules generally will be stored in some logical hierarchical sequence, it is not strictly necessary. You could put the rules in any sequence and the inference engine would find them in the right order when it goes to solve a problem.

Inference Engine

The inference engine is the algorithm that controls the reasoning process. Also called a "control program" or "rule interpreter," the program works with the input data supplied by the user to search the knowledge base in order to reach a conclusion. Its control strategies implement either forward- or backward-chaining, depth-first or breadth-first search, or a combination of these. The pattern-matching function of the inference engine compares the input data to either the IF or THEN portion of rules in the knowledge base. When a match is found, one portion of the reasoning process is complete. The inference engine continues its search until it provides a solution.

The inference engine is the main controller in an expert system as it implements the search.

The inference engine runs the whole show. Its two basic functions are inference and control. Inference refers to examining the rules and performing the pattern-matching while control refers to the sequence in which rules are examined. The inference engine asks the user for initial input, then it proceeds to search through the rule base looking for rules that match the input. The rule interpreter determines the sequence in which it examines rules, and it asks for additional input information if it cannot make a decision based on the facts it has and the rules available. The inference engine fully automates this process and it is totally invisible when running a consultation.

User Interface

The user interface facilitates user-software communications.

The user interface is a collection of programs that work with the inference engine and the knowledge base to provide a convenient means of two-way communications with the user. The user interface gathers input data in one of two ways. The expert system may ask questions to which the user replies by typing in answers. Or the user interfaces may operate by menus. These offer multiple-choice questions, asking the user to select the correct choice from among several alternatives. This makes data entry fast and simple.

The user interface also comes into play during the inferencing process. Should the rule interpreter find that it cannot reach a decision because it has insufficient information, it notifies the user via the monitor. It may do this by picking up a portion of the rule being tested and reformat it into a question. When the user enters the data asked for, the inference engine can continue its reasoning process.

When the inferencing process is complete an output result is presented on the monitor. Expert systems cannot always reach the right answer, so the screen output may state that no conclusion or only a guess or estimate of the output could be reached. Or the result may be an answer, but with a certainty factor qualifying it.

Simple user interfaces take the verbiage built into the rule base and reformat or resequence it as inputs and outputs. If the rules are written clearly in English, the inputs and outputs should be relatively easy to understand. There is a temptation when creating an expert system to use abbreviations and shorthand notations to minimize the size of rules, but doing so makes the program more difficult to understand.

An expert system may use a natural language front-end or interface. This program uses AI techniques to improve communications between user and expert system. The natural language front-end understands English language inputs and interfaces with the rest of the system to interpret input and convert it to an internally usable form. In the same way, conclusions from the system are reformatted into conversational English. Most expert systems don't use natural language front-ends because they are often as difficult to construct as the entire expert system itself. Generally, only on very large systems can the expense of a natural language front-end be justified.

Data Base

The data base is the expert system's bulletin board for sharing information and status internally.

The data base, sometimes called the global data base or working memory, is the portion of the computer's memory set aside for keeping track of inputs, intermediate conclusions, and outputs. The inference engine uses the data base like a scratch pad to track what's going on in the system.

Initial inputs are stored in the data base. As the rule interpreter sequences through the rules, the conclusions drawn from each of those rules are stored in the data base. The inference engine uses these intermediate conclusions as new inputs to search for new matches. At the end of a run, the data base contains the entire chain of facts that include not only those entered initially but also those that were concluded along the way to the final decision.

Explanation Subsystem

Many expert systems contain a section designed to explain to the user what line of reasoning was used to reach its conclusion. Many users find it difficult to trust the advice of an expert system, not unlike the distrust they might feel for the advice of an unfamiliar human consultant.

Users put more trust in an expert system decision when they can understand its line of reasoning.

One way to solve this problem is to have the system explain how it reached its conclusion. With an explanation subsystem, users can ask "Why?" or "How?" and the system can give an answer. For example, if the system asks for additional input data, the user might wish to ask why. Usually, the system would respond by saying it needs the information to evaluate a particular rule and it might even show which rule it is attempting to satisfy.

Asking how usually causes the system to show the full sequence of rules it examined in order to reach its conclusion. By seeing the logical reasoning process, users can better accept the outcome. Some larger systems annotate the rules and otherwise provide expanded discussion and justification.

The explanation subsystem is also an excellent feature for instructional purposes. Most expert systems quickly provide an answer to a problem, but typically they don't explain their reasoning unless asked. Through solving a number of problems and at the same time asking for an explanation on each, users may begin to understand the reasoning process. With enough practice, users might become experts in their own right.

Users can also use the explanation subsystem for debugging purposes. During development, it can serve as a way to get feedback on rule construction and sequence, enabling users to readily test the system on practical problems.

This capability of explanation by an expert system is often a valuable feature where critical decisions are being made. If users can question the decisions of the system and explore alternatives, they bring their own knowledge to bear on the problem and make the conclusion a thoroughly considered joint decision. For example, a doctor using an expert system to diagnose an illness may want to understand the logical process used by the system and may second-guess its diagnosis.

But not all applications require nor warrant an explanation capability. For example, a fighter pilot using an expert system to help make critical, split-second maneuvering or weapons-choice decisions certainly doesn't have time to question or analyze a decision.

Knowledge Input Subsystem

An expert system needs a way to acquire any additional knowledge it needs to function.

Most expert systems contain a program or set of programs that enables users to add to or modify the rule base. Many expert system domains are dynamic in nature. That is, the knowledge is constantly changing and, therefore, the rule base must be modified to reflect these changes. The knowledge acquisition subsystem provides a convenient means of adding new rules and editing existing rules. While the subsystem is usually a specialized text editor, a standard word processing software package may be used in some systems. In this case, additional rules or rule revisions are keyed in on a word processor and a file created. The knowledge acquisition subsystem reads this information and performs the knowledge base update, acting like a compiler converting the text into the correct format.

Maintaining an expert system is very important. If domain knowledge changes frequently, the expert system begins providing wrong answers if the new knowledge is not added. The knowledge acquisition subsystem provides a fast and convenient way to make these important changes.

EXPERT SYSTEM OPERATION

Now let's see how an expert system actually goes about arriving at a decision. Ours will be a simple rule-based expert system that can perform either forward- or backward-chaining. We'll supply initial inputs and see the actual sequence of rule examinations that the inference engine uses to arrive at a conclusion. Along the way, we'll examine how certainty factors play a role in this conclusion.

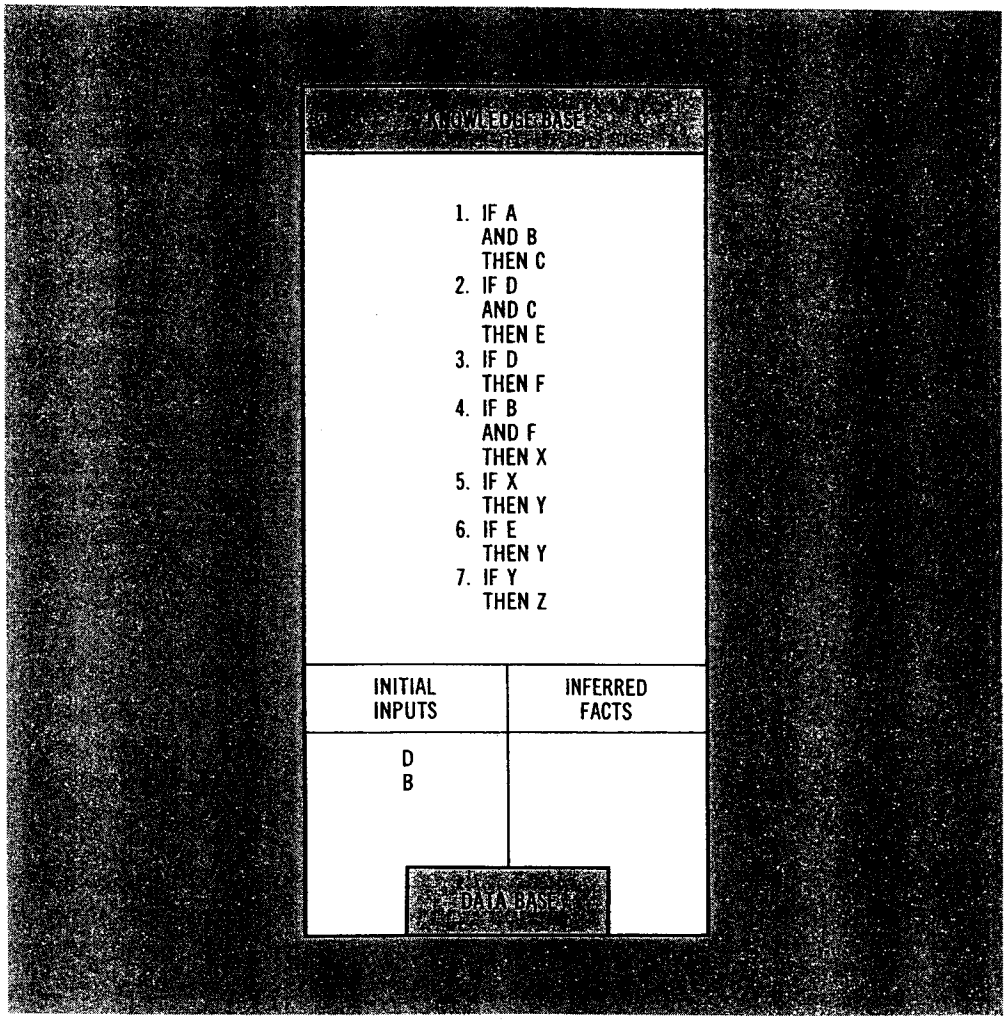
Example Knowledge Base

Figure 5-2 shows a seven-rule knowledge base. Instead of using factual statements, we are using symbols—letters of the alphabet—to represent our premise and conclusion statements of heuristic knowledge. In this way, it will be easier to follow the operational sequence. Later, we'll analyze some real expert systems to further explain this process.

Example Data Base

The figure also shows our expert system data base. Note that the initial inputs B and D have been stored there. These facts are the starting point for the inference engine.

Figure 5-2. Hypothetical Knowledge Base and Data Base



The other part of the data base is allocated to new facts that will be inferred as the inference engine proceeds to analyze the rules. Whenever a rule is satisfied, a new conclusion is drawn and put into the data base. These intermediate conclusions ultimately lead to our final answer.

Inference Engine Operation

The control strategy built into the inference engine determines how the rules in the knowledge base will be examined. Usually, the inference engine, or "rule interpreter" as may be more appropriate in this case, is usually set up to implement either a forward-chaining or a backward-chaining control sequence. We'll take a look at both methods here to be sure you understand how they work.

In both cases the inference engine looks at each rule in a particular sequence and attempts to infer new information. The inference engine will use the initial facts stored in the data base and attempt to match them with the IF or THEN statements in the rules it is examining. We saw in Chapter 4 that if all conditions of a rule are satisfied, the rule is said to fire. For example, in our rule number 1 in *Figure 5-2*, should both conditions A and B be known, then C is inferred and rule 1 *fires*. This means that the designated conclusion is true and the action it expresses is carried out. A rule fires only when all of its conditions are satisfied. When a rule fires, the conclusion drawn is stored in the data base so that it may be used by the inference engine to seek matches in other rules.

Every time a new rule is examined, the inference engine checks the data base to determine what facts it knows and attempts to match them to the new rule. If insufficient information is available to satisfy a rule, the expert system may ask for new inputs. Otherwise, the inference engine simply moves on to the next rule in sequence, again attempting to match what it knows in the data base to the conditions in the rule itself. The inference engine will continue until it runs out of rules and facts. At that point, the expert system usually presents its conclusion.

Backward-Chaining

In backward-chaining, one of the possible outcomes is selected as a hypothesis and the rule base is searched for its proof.

Most rule-based expert systems use backward-chaining in their inferencing process. In backward-chaining, the expert system attempts to prove an hypothesis. Should all of the facts available in the data base and those inferred by the inferencing process prove the hypothesis, then it is assumed to be true and that becomes the output recommendation. All expert systems have stored in them the conclusions that are possible for a given set of problems in the domain. In creating the knowledge base, the expert and the knowledge engineer have to define all possible outcomes. The knowledge base contains a rule that will infer each of those conclusions. The other rules in the knowledge base provide intermediate inferencing steps that will ultimately lead to one of the rules proving the hypothesis. With backward-chaining, the expert system chooses one of these final conclusions as an hypothesis and then attempts to prove it given the input data. Should it not be possible to prove the hypothesis, the inference engine goes on to attempt to prove the next outcome. Sooner or later, the expert system will prove one of the hypotheses and the session will be over.

Refer again to the knowledge base and data base shown in *Figure 5-2*. In this expert system, our hypothesis is Z. For simplicity, we have only one outcome that can be proven. The inference engine begins by looking for Z in the data base to see if it has already been proven. Z is not there. In backward-chaining, the hypothesis is compared to the THEN parts of the rules until a match is achieved. So the inference engine begins its search by looking for rules whose outcome is Z. Only rule 7 matches Z. Now the inference engine needs to prove Y in order for rule 7 to be true and Z to be concluded.

The inference engine looks for Y in the data base. Since it is not there, the inference engine searches for rules that conclude Y. Rules 5 and 6 meet this criteria. The inference engine looks at rule 5 first. To satisfy rule 5, X must be true. X is not in the data base, so the inference engine begins to look for rules that conclude X. Rule 4 is identified but rule 4 requires that both B and F be known if it is to be satisfied.

The inference engine looks in the data base again and finds that condition B is met. B is one of the initial facts we entered. Next, the inference engine looks for F. F is not in the data base so the inference engine begins to look for a rule that concludes F. It discovers rule 3. In order for F to be true, D must be true. The rule interpreter looks for D in the data base and finds it. It too was entered as one of our initial facts.

With this last operation, our hypothesis has been proven. You can see now that rule 3 is satisfied because fact D was initially supplied and, therefore, it concludes F. Rule 4 needs both B and F to conclude X. F is true and B was initially supplied so X is true and rule 4 fires. Since X is now true, rule 5 fires, inferring Y. With Y now known, rule 7 fires, concluding Z, our original hypothesis. Our data base is now shown in *Figure 5-3*.

Assume that the inference engine examines rule 6 instead of rule 5 in attempting to satisfy rule 7. Rules 5 and 6 both conclude Y and either will satisfy the condition in rule 7. If the rule interpreter looks at rule 6, it needs to find E in order to fire. E is not in the data base, so the inference engine looks for a rule that concludes E. It discovers rule 2. In order for rule 2 to fire, facts C and D must be known. The inference engine finds fact D in the data base because it was one of the initial input facts, but it doesn't find C in the data base. It does, however discover rule 1 which infers C. The inference engine then looks for A and B to satisfy rule 1. It finds fact B in the data base but not fact A. A search through the rule base finds that none of the rules conclude A. At this point, the expert system would communicate with the user and ask for

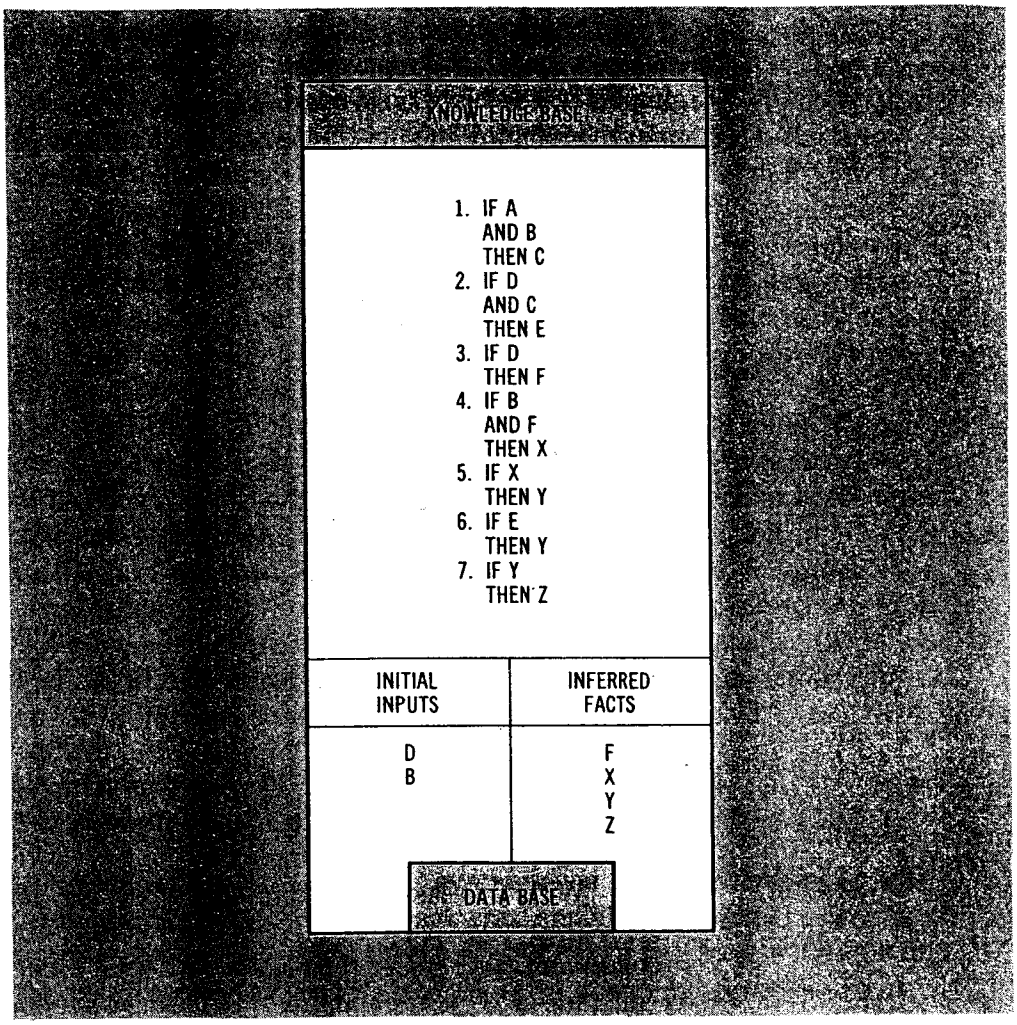


Figure 5-3. Hypothetical Knowledge and Data Base, Inferred Facts Added

additional input. Should the user be able to supply fact A, then a new sequence of hypothesis-proving would occur. However, if A is not known, the system would try another route to prove the original hypothesis Z. In this case, the system would backtrack and discover that rule 5 also concludes Y and then take the path previously described.

There are two key points to remember about backward-chaining. First, the inference engine attempts to prove one of the conclusions that it already knows. These conclusions are tested one after another based on given information. With sufficient input data, one of the outcomes will be proven.

Second, note that in backward-chaining the inference engine looks at the THEN part of the rule first and then attempts to prove the IF portion. It looks in its data base for rules that conclude that portion of the IF statement. If none of those are possible, it asks

the user to supply the necessary input. Should the necessary input not be available, the inference engine looks for other rules or concludes that the particular hypothesis under test cannot be proven at all. It then moves on to the next hypothesis and each additional hypothesis in sequence until one is proven. Of course, if there is insufficient input data, the system simply cannot supply a conclusion.

Forward-Chaining

In forward-chaining, input facts are matched to rule IF statements until a solution is reached.

If a forward-chaining control sequence is implemented, the inference engine will start with any available facts in the data base and search for those facts in the IF portions of the rules. If the IF part of the rule matches a fact in the data base, the rule is fired. The THEN portion of the rule is said to be true and a new fact is thus inferred and stored in the data base. With this new information, the inference engine moves forward to find this newly inferred fact in the IF part of another rule. This process continues until no further conclusions can be reached. At that point, the expert system has its answer.

To show you how forward-chaining works, we'll begin with the rule base and data base given in *Figure 5-2*. Ultimately, of course, the expert system will conclude that outcome Z is true.

To begin, the inference engine goes to the data base looking for initial facts. It finds that it knows B and D. Therefore, it takes the first one, B, and begins searching rules for IF statements that match. It looks at rule 1 and sees B, but A is also required to fire this rule. A is not in the data base. It moves on to rule 2 and finds no B. Moving on to rule 3, the inference engine finds no B. Rule 4 is examined next. B is found but F, which isn't in the data base, is also required. Rules 5, 6, and 7 are examined without success.

The inference engine backtracks, picking up D from the data base. It looks at rule 1 without a match. Rule 2 contains D but C is not available. Rule 3 contains D and it fires. Thus F is concluded and stored in the data base.

The remaining rules are again tested without success. A conclusion still has not been reached so the inference engine backtracks again. Again it searches for B. The previously described sequence takes place until rule 4 is encountered. The IF statement of rule 4 contains B so a match is found. But rule 4 also requires F to satisfy it. The inference engine looks in the data base for F and finds it there. Recall that F was concluded when rule 3 fired. The conditions for rule 4 are met and, therefore, rule 4 fires, concluding that X is true. X is stored in the data base.

An inference engine backtracks until it finds an answer.

The inference engine has used up its initial inputs and now goes on to the new facts that are stored there. These new facts, F and X, were inferred from rules that fired. The rule interpreter first looks for a match to F in the IF portions of rules and finds rule 4, but it has already fired and, therefore, no further action is taken. The inference engine moves on to fact X and begins looking for rules whose IF statements match. It discovers rule 5 which fires and immediately infers Y. Y is added to the data base: The inference engine continues its attempt to match Y to other IF statements in the rule base. It finds rule 7 and, therefore, it fires, concluding Z. Thus the original hypothesis is proven. *Figure 5-4* shows a decision tree whose nodes are the facts and whose arcs represent the rules that fire to conclude those facts.

Figure 5-4.
Decision Tree Showing Forward-Chaining Path